

AMERICAN UNIVERSITY OF BEIRUT

Parallel Time Methods for Computing a  
Satellite's Trajectory

by

SAMAH WAHID KARIM

A thesis

submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Computational Science  
of the Faculty of Arts and Sciences  
at the American University of Beirut

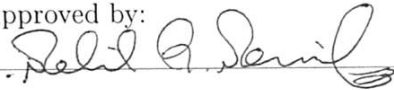
Beirut, Lebanon  
May 2014

# AMERICAN UNIVERSITY OF BEIRUT

## Parallel Time Methods for Computing a Satellite's Trajectory

by  
SAMAH WAHID KARIM

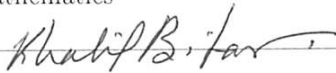
Approved by:



Dr. Nabil Nassif, Professor

Advisor

Mathematics



Dr. Khalil Bitar, Professor

Member of Committee

Physics



Dr. Leonid Klushin, Professor

Member of Committee

Physics

Date of thesis defense: May 8<sup>th</sup>, 2014



# Acknowledgements

I would like to express my gratitude to my advisor Professor Nabil Nassif who has been a mentor to me for many years now. I would like to thank him for introducing me to Computational Science and encouraging me to pursue graduate studies in this field. Thank you for proposing this problem for my Master's thesis and for guiding me all along the way.

I also want to express my appreciation to the Committee members Professor Khalil Bitar and Professor Leonid Klushin for their useful feedback and discussions.

I want to thank my fiancé Mohammad Nouredine for being my companion on this journey. Thank you for being my rock and my anchor. Thank you for your unconditional love and support. I could not have done this without you.

I also want to thank my family, my father Wahid, my mother Mariam, my brothers Mohammad and Ali, for believing in me and encouraging me to pursue my dreams. A special thank you to Mariam who has always been my number 1 fan.

Last but not least, I cannot but thank God for absolutely everything. Thank you for giving me my wonderful fiancé and family. Thank you for guiding me to find a field that I am very passionate about: Computational Science.

# AN ABSTRACT OF THE THESIS OF

SAMAH WAHID KARIM for Master of Science  
Major: Computational Science

Title: Parallel Time Methods for Computing a Satellite's Trajectory

Solving time dependent ordinary differential equations in a time-parallel way was thought to be impossible since time integration is inherently sequential. However in recent years, several predictor-corrector schemes have been proposed in order to solve time dependent differential equations. The most prominent of these algorithms is the well known Parareal algorithm (Lions et al 2001, Gander et al 2007) and more recently the Adaptive Parallel Time Integration algorithm "APTI"(Nassif et al 2005). Such method has been successfully applied to the problem that models the motion of a membrane element linked to a spring (Karam, Nassif, Erhel 2013).

In this thesis, we implement APTI in order to calculate the trajectory of a satellite, governed by a perturbed Keplerian model. We present also a modified version of the Parareal algorithm that utilizes the convergence of some slices prior to the global convergence, in order to enhance execution time. In order to test the efficiency of APTI, we compare its results with those obtained by Parareal and Modified Parareal. We show speed-ups as well as deviation from trajectories computed on the basis of a sequential algorithm, indicating the efficiency of the APTI approach for relatively large periods of time.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Satellite Trajectory</b>	<b>3</b>
2.1	Derivation of a Simplified Satellite Model . . . . .	3
2.2	Reference Coordinate Systems . . . . .	7
2.2.1	Classical Coordinate Systems . . . . .	7
2.2.2	Our Coordinate System: Initially Perifocal Coordinate Frame (IPQW-Frame) . . . . .	10
2.3	Variables of the Motion . . . . .	10
2.4	Coordinate Transformations . . . . .	11
2.5	$J_2$ -perturbed Motion as a System of ODE's . . . . .	16
2.5.1	Expression of $\vec{\nabla}U$ in the ECI-frame . . . . .	16
2.5.2	Equivalent System of First Order ODE's . . . . .	18
<b>3</b>	<b>Adaptive Parallel Time Integration</b>	<b>21</b>
3.1	Sequential Solvers . . . . .	21
3.2	Parallel Solvers . . . . .	23
3.2.1	Time Parallelism . . . . .	24
3.3	<u>A</u> dpative <u>P</u> arallel <u>T</u> ime <u>I</u> ntegration . . . . .	26
3.3.1	Automatic Generation of Time-Slices: End Of Slice (EOS) Condition . . . . .	27
3.3.2	Selection of an EOS condition . . . . .	28
3.3.3	Ratio vectors and Ratio property . . . . .	30
3.3.4	Change of variables . . . . .	32
3.3.5	Ratio-Based Prediction and Correction Procedures . . . . .	38
3.4	APTI Algorithm . . . . .	39
3.5	Increase Speed-Up through a Duplication Approach . . . . .	42

<b>4</b>	<b>The Parareal Algorithm: Parallel in Real Time</b>	<b>45</b>
4.1	Related Work . . . . .	45
4.2	LIONS Parareal Algorithm . . . . .	47
4.3	Implementation . . . . .	50
4.4	APTI vs Parareal . . . . .	53
4.5	Modified Parareal . . . . .	55
<b>5</b>	<b>Results</b>	<b>58</b>
5.1	Choice of an EOS condition for the APTI algorithm . . . . .	59
5.2	Period of 1 day . . . . .	61
5.3	Period of 3 days . . . . .	69
5.4	Period of 20 days . . . . .	72
5.5	Period of 43 days . . . . .	75
5.6	Period of 108 days . . . . .	78
<b>6</b>	<b>Conclusion</b>	<b>81</b>

# List of Figures

2.1	Earth Centered Inertial Coordinate Frame (ECI) . . . . .	8
2.2	Perifocal Coordinate Frame (PQW) . . . . .	9
2.3	Satellite Orbital Elements (a, e, E, $\theta$ ) . . . . .	12
2.4	Satellite Orbital Elements (i, $\omega$ , $\Omega$ ) . . . . .	12
3.1	Multiple shooting method . . . . .	25
3.2	Solving rescaled systems locally . . . . .	34
3.3	Cyclic Distribution of Slices Among 4 Processors . . . . .	42
3.4	Gaps Between Calculated and Predicted Values of the Solution	43
3.5	Classical versus Duplication Approach for APTI . . . . .	44
4.1	Initial guess of the solution of the Brusselator problem . . . . .	51
4.2	Final approximation of the solution of the Brusselator problem	51
4.3	Solution of the Brusselator Problem given by the Parareal and Modified Parareal algorithms . . . . .	57
5.1	Orbit of a satellite in a $J_2$ perturbed motion (11 rotations) . .	60



# List of Tables

4.1	Performance results for the Brusselator problem . . . . .	57
5.1	Notations used . . . . .	58
5.2	Performance results for case 1 initial conditions for 1 day . . .	62
5.3	Performance results for case 2 initial conditions for 1 day . . .	63
5.4	Performance results for case 3 initial conditions for 1 day . . .	64
5.5	Performance results for case 4 initial conditions for 1 day . . .	65
5.6	Performance results for case 5 initial conditions for 1 day . . .	66
5.7	Performance results for case 6 initial conditions for 1 day . . .	67
5.8	Average performance results for a period of 1 day . . . . .	68
5.9	Speed up of Modified Parareal versus Parareal for a period of 1 day . . . . .	68
5.10	Performance results for case 1 initial conditions for 3 days . .	69
5.11	Performance results for case 2 initial conditions for 3 days . .	70
5.12	Average performance results for a period of 3 days . . . . .	71
5.13	Speed up of Modified Parareal versus Parareal for a period of 3 days . . . . .	71
5.14	Performance results for case 1 initial conditions for 20 days . .	72
5.15	Performance results for case 2 initial conditions for 20 day . .	73
5.16	Average performance results for a period of 20 days . . . . .	74
5.17	Speed up of Modified Parareal versus Parareal for a period of 20 days . . . . .	74
5.18	Performance results for case 1 initial conditions for 43 days . .	75
5.19	Performance results for case 2 initial conditions for 43 days . .	76
5.20	Average performance results for a period of 43 days . . . . .	77
5.21	Speed up of Modified Parareal versus Parareal for a period of 43 days . . . . .	77
5.22	Performance results (1) of APTI for 108 days . . . . .	78
5.23	Performance results (2) of APTI for 108 days . . . . .	79

5.24 Average performance results of APTI for a period of 108 days 80

# Chapter 1

## Introduction

In space missions, one needs to account for the orbit of satellites whether natural or manmade. Thus one needs to solve a system of second order differential equations for the orbit of the satellite and needs to be updated about its solution, all along the mission (see [10],[9]). In order to do so, it is required to perform a mass of expensive computations due to the many elements that need to be taken into account.

In that context and given the complexities that the mathematical model may reach, one needs to devise a way through which the time consumption of the required computations is reduced. Usually, integration of a time-dependent system of ordinary differential equation is by default sequential in nature. However in recent years due to the development of parallel computer architectures with thousands of processors, several authors have proposed parallel schemes to solve a time-dependent differential equation. In 1998, S. Rault proposed a multiple shooting method to solve this sort of equations in a parallel way [9].

More generally in 2001, Lions, Maday and Turinici proposed a new approach to solve in parallel time-dependent problems in [23]. Their “parareal algorithm” received great attention and has been subject to many implementations and developments, as in [4],[11],[25],[24],[30]. Other contributions to this algorithm can also be found in [32], [3], [13], [17], [31], [32].

As the parareal algorithm uses a restrictive regular coarse grid, another

parallel approach to solve time-dependent problems has been also devised in [27] and [22]. It is based on a sliced-time approach found in [26] and was successfully implemented to unbounded solutions appearing in reaction diffusion problems and oscillating membranes. One of the main features of the method is its “adaptive” character. Specifically, by allowing for automatic generation of non-uniform coarse grids, it reduces the time of executing sequential processes necessary to predict solution values on future periods of times. Such method has been referred to as “Adaptive Parallel Time Integration (APTI)”.

In this thesis, we propose to apply APTI to a simplified model of the satellite problem. We also implement the Parareal algorithm to the problem and compare the results given by the above procedures.

This thesis is organised as follows. In chapter 2, we describe the satellite trajectory problem and derive the key equations that need to be solved for the simplified  $J_2$ -model. In chapter 3, we describe the APTI algorithm and the main steps involved in it. In chapter 4, we discuss the Parareal algorithm and compare it with the APTI algorithm. In chapter 5 we present the results of applying both algorithms to the satellite problem, for different initial conditions and integration periods. In chapter 6, we conclude and suggest future work.

# Chapter 2

## Satellite Trajectory

### 2.1 Derivation of a Simplified Satellite Model

Our main reference for the analysis of the satellite trajectory problem is that of O.Zarrouati [34].

**General Equation of a Satellite's Trajectory:**

Newton's laws govern the motion of the satellites in space. In fact Newton's second law is the starting classical mechanical law used to obtain a mathematical model for the satellite's orbit:

$$\vec{F} = m\vec{r} \tag{2.1}$$

where:

$m$  is the mass of the satellite,

$\vec{r}$  is the position-vector of the satellite with respect to the center of the earth

$\vec{r}$  is the acceleration vector of the satellite, and

$\vec{F}$  is the force vector applied on the satellite.

**Keplerian Motion:**

The central gravitational attraction is considered the only force that is ap-

plied to the satellite in the proximity of the earth:

$$\vec{F} = -GM \frac{m}{\|\vec{r}\|^3} \vec{r} = -\mu \frac{m}{\|\vec{r}\|^3} \vec{r} \quad (2.2)$$

where  $G$  is the universal gravitational constant,  $M$  the earth's mass and

$$\mu = GM = 3986005 \times 10^8 m^3/s^2. \quad (2.3)$$

Then, equation (2.1) reduces to the following system of differential equations:

$$\ddot{\vec{r}} = -\frac{\mu}{\|\vec{r}\|^3} \vec{r}. \quad (2.4)$$

- Kepler's first law states that the satellite moves in a fixed plane along an ellipse having the center of the earth as one focus. One can define this "Keplerian plane" using the initial position and velocity vectors of the satellite solely.

If  $a$  and  $b$  denote the semi-major and semi-minor axes respectively, then the eccentricity of the elliptical orbit is  $e = \frac{\sqrt{a^2 - b^2}}{a}$ , and its semi-latus rectum is  $p = \frac{b^2}{a} = a(1 - e^2)$ . The polar equation of the ellipse is given by:

$$r = \frac{p}{1 + e \cos \theta},$$

where  $r$  and  $\theta$  are the polar coordinates from the center of the earth to the satellite and  $\theta = 0$  along the major axis directed toward the perigee,

- Kepler's second law states that the satellite revolves so that the line joining it to the center of the earth sweeps out equal areas in equal time intervals. It implies:  $r^2 \dot{\theta} = \sqrt{\mu a(1 - e^2)}$ ,
- Kepler's third law states that the motion of the satellite is periodic of

period:

$$P = 2\pi \sqrt{\frac{a^3}{\mu}}. \quad (2.5)$$

It gives:  $n^2 a^3 = \mu$ , where  $n = \frac{2\pi}{P}$  is the mean motion of the satellite (i.e. mean angular velocity) and  $P$  is the period of the the satellite.

### **Perturbing Forces:**

But in actuality, the orbit of the satellite is not Keplerian as many other forces perturb the central gravitational attraction.

Some of these perturbations are surface forces for instance solar pressure, friction force, while others result from gravitational potentials of other planets, of the earth itself, of other satellites.

Rault detailed these forces in [9] and compared their order of magnitude to that of the central attraction of the earth as follows:

<i>Solar pressure</i>	:	$10^{-9}$
<i>The sun's gravitational attraction</i>	:	$10^{-8}$
<i>The moon's gravitational attraction</i>	:	$10^{-7}$
<i>Atmospheric friction</i>	:	$10^{-9}$ to $10^{-5}$
<i>Force due to the flattening of the earth</i>	:	$10^{-3}$
<i>Central gravitational attraction of the earth</i>	:	1

In this thesis, we will only consider the perturbative force caused by the flattening of the earth since it is by far the most significant perturbation to the motion of the satellite.

However, the algorithms we will be using to solve this problem, can account for other perturbing forces.

### **General Gravitational Potential of The Earth:**

The gravitational potential of the earth can be expanded into a series of spherical harmonics:

$$U = -\frac{\mu}{r} \left\{ 1 + \sum_{n=1}^{\infty} \left(\frac{r_{eq}}{r}\right)^n J_n P_n(\sin \varphi) + \sum_{n=1}^{\infty} \sum_{m=1}^n \left(\frac{r_{eq}}{r}\right)^n [C_{nm} \cos(m\lambda) + S_{nm} \sin(m\lambda)] P_{nm}(\sin \varphi) \right\} \quad (2.6)$$

where  $r$  is the magnitude of the position vector  $\vec{r}$ ,  $r_{eq}$  the equatorial radius,  $\varphi$  the geocentric latitude,  $\lambda$  the longitude,  $P_k$  the Legendre polynomials of degree  $k$ ,

$P_{n,m}$  the Legendre functions of degree  $n$  and order  $m$ ,  
 $J_n$  the zonal harmonics of order  $n$  (with  $J_1 = 0$ ),  
 $S_{nm}$  and  $C_{nm}$  the tesseral harmonics of degree  $n$  and order  $m$ .

The earth's flattening expresses itself in the zonal harmonic  $J_2$  which makes the earth shaped like an ellipsoid rather than a sphere.

But in fact the earth is shaped like a pear rather than a perfect ellipsoid. This pear shape is due to the spherical harmonic  $J_3$ .

One notes that only zonal harmonics such as  $J_2, J_3, J_4, \dots$  will cause significant changes to the orbit of the satellite.

And amongst the above mentioned, the zonal harmonic  $J_2$  has the the biggest contribution to the deviation of the earth's force field from that of a sphere.

### **The $J_2$ -Perturbed Gravitational Potential of The Earth:**

Therefore, the  $J_2$ -model for the satellite trajectory takes into consideration the central gravitational attraction of the earth along with the most dominant perturbation, which is due to the flattening of the earth, while neglecting all other perturbations forces.

According to this model, the gravitational potential of the earth is expressed as:

$$u(J_2) = -\frac{\mu}{r} \left\{ 1 + \left( \frac{r_{eq}}{r} \right)^2 J_2 P_2(\sin \varphi) \right\}, \quad (2.7)$$

where:

$\mu = 3986005 \times 10^8 m^3/s^2$ , as given in (2.3),  
 the equatorial radius  $r_{eq}$  is equal to:

$$r_{eq} = 6378.137 km, \quad (2.8)$$

the zonal harmonic  $J_2$  is:

$$J_2 = -11.10^{-4}, \quad (2.9)$$

$r = \|\vec{r}\|_2$  is the norm of the position vector  $\vec{r}$ ,  
 $P_2(\sin \varphi)$  is the Legendre polynomial of degree 2, in the variable  $\sin \varphi$ , where



$\varphi$  is the latitude:

$$P_2(\sin \varphi) = \frac{3}{2} \sin^2 \varphi - \frac{1}{2}, \quad (2.10)$$

**Principal Differential Equation for the  $J_2$ -model:**

The force per unit mass deriving from the potential  $u(J_2)$  (2.7) is equal to  $-\vec{\nabla}u(J_2)$ . Therefore the force applied to the satellite is expressed as:  $\vec{F} = -m\vec{\nabla}u(J_2)$ , and the general equation of motion (2.1) reduces to a second-order differential equation:

$$\vec{\ddot{r}} = -\vec{\nabla}u(J_2) \quad (2.11)$$

In order to fully determine the trajectory of the satellite, one needs to specify a set of initial conditions.

## 2.2 Reference Coordinate Systems

### 2.2.1 Classical Coordinate Systems

Two classical coordinate systems that are usually used for this problem are presented below.

- **Earth Centered Inertial Coordinate Frame (ECI):**

This frame is a cartesian coordinate system  $(O, \vec{i}, \vec{j}, \vec{k})$ , which is centered on the earth and where:

- The X-axis goes from the center of the Earth through the earth’s equator at the “the vernal equinox”, also called “the first point in Aries”. This point is where the sun crosses the equator of the great celestial sphere, going north. We can also define this axis to be the intersection between the Earth’s equatorial plane and the ecliptic plane(the plane orthogonal to the Earth’s axis of rotation),
- The Z-axis is parallel to the Earth’s axis of rotation, and

- The Y-axis is obtained by applying the right-hand rule, i.e.  $\vec{j} = \vec{k} \times \vec{i}$ .

These axes are fixed in space, therefore all the points on Earth, except for the poles, rotate  $360^\circ$  around the z-axis of the ECI frame every 24 hours. Moreover, the origin of this coordinate frame which is the center of the Earth, is not actually fixed in space, since the Earth is moving around the Sun. Therefore, the ECI frame is a moving coordinate system and thus is not inertial (“fixed”). That is why it is called an Earth Centered Inertial coordinate system. Note that it can be considered truly inertial in the proximity of the Earth. For the purpose of the problem we are dealing with, and since satellites that orbit the Earth are always near the Earth, we can treat this coordinate frame as truly fixed in space without loss of accuracy.

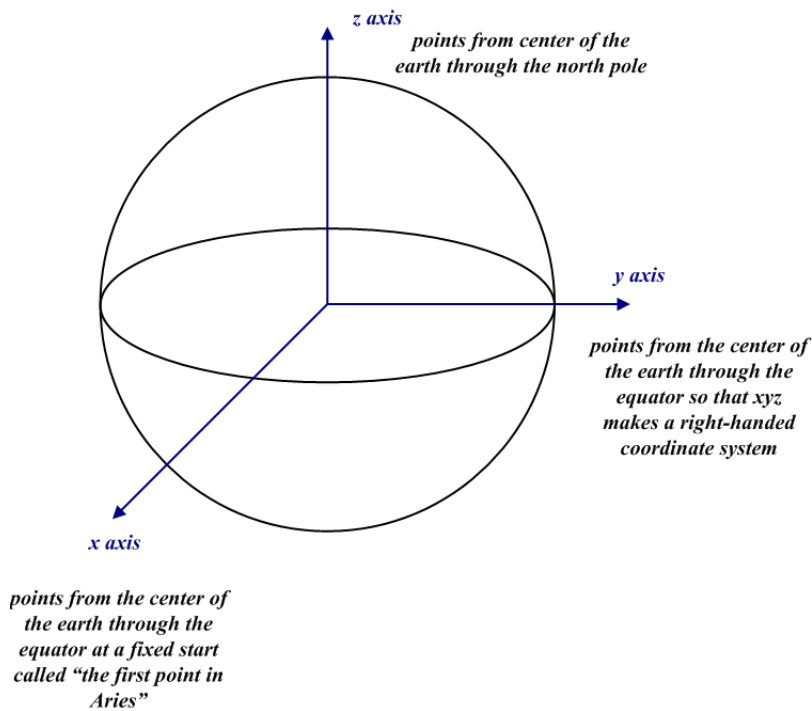


Figure 2.1: Earth Centered Inertial Coordinate Frame (ECI)

- **Perifocal Coordinate Frame (PQW):**

This is a cartesian coordinate system  $(O, \vec{P}, \vec{Q}, \vec{W})$  that is based on both the Earth and the satellite, where:

- the X-axis  $(O, \vec{P})$  points from the center of the Earth toward the location of the perigee, which is the closest point on the elliptical orbit to the center of the Earth,
- the Y-axis  $(O, \vec{Q})$  points from the center of the Earth in a direction that is orthogonal to the X-axis in the Keplerian elliptical plane,
- the Z-axis  $(O, \vec{W})$  points from the center of the earth in a direction that is orthogonal to the elliptical orbital plane, following the right hand rule for the triad  $\vec{P}, \vec{Q}, \vec{W}$ , i.e.  $\vec{W} = \vec{P} \times \vec{Q}$ .

One ought to note that in a Keplerian motion, the PQW-frame is an inertial reference frame since the elliptical orbit is fixed in space. But when the motion is perturbed, the orbital plane will be moving. The PQW-frame is then defined instantaneously by  $(\vec{r}(t), \dot{\vec{r}}(t))$ , so that  $\vec{P}$  and  $\vec{Q}$  lie in the instantaneous elliptical plane, i.e. the resultant ellipse if all perturbations would cease at that instant, while  $\vec{W}$  is normal to it.

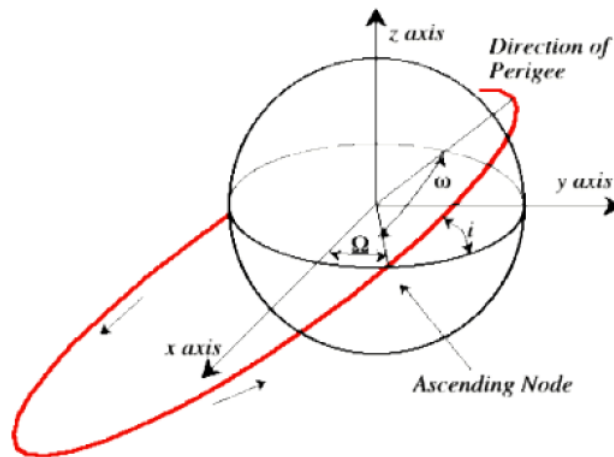


Figure 2.2: Perifocal Coordinate Frame (PQW)

## 2.2.2 Our Coordinate System: Initially Perifocal Coordinate Frame (IPQW-Frame)

If the gravitational attraction of the earth was centered, then the gravitational potential would have been reduced to its first term  $\mu/r$  and the satellite would have moved, in a Keplerian motion, in a fixed plane called the Keplerian plane, which is completely defined by  $\vec{r}_0$  and  $\vec{V}_0$ .

However when a perturbation is added, the satellite will not follow a closed elliptic path anymore. Instead it will follow a trajectory that is always tangential to an instantaneous ellipse, which is non other than the osculating ellipse. This ellipse is defined by the instantaneous values of the orbital elements. This means that the perturbed physical trajectory would coincide with the Keplerian orbit that the satellite would follow if the perturbing force was to cease instantaneously.

Since the osculating ellipse does not remain constant, it is convenient to use a fixed *reference orbit* - for instance, the osculating ellipse at the initial time  $t_0$ .

Thus we introduce a new coordinate frame, the **IPQW-frame**, which is the **PQW-frame corresponding to the initial conditions** and keep it fixed with respect to time.

## 2.3 Variables of the Motion

To fully describe the trajectory of the satellite, 6 independent variables are needed. There are 2 choices for these variables: either the classical Cartesian coordinates or the Keplerian orbital elements [19]. Normally, analytic analysis of this problem tend to always use the Keplerian orbital elements, while numerical analysis tend to use Cartesian variables.

Note that in our computations, we will use the orbital elements for the sole purpose of defining the initial conditions.

- **Cartesian variables:**

The set of Cartesian parameters  $(x, y, z, \dot{x}, \dot{y}, \dot{z})$ , made of the components  $(x, y, z)$  of the position vector  $\vec{r}$  and the components  $(\dot{x}, \dot{y}, \dot{z})$  of the velocity vector  $\vec{v}$ .

- **Orbital Elements:** The set of Keplerian orbital elements  $(a, e, i, \omega, \Omega, M)$ , where:
  - $a$ : semi-major axis of the elliptical orbit,
  - $e$ : eccentricity of the elliptical orbit,
  - $i$ : inclination of the orbit, i.e. the angle between the orbital plane and the earth's equatorial plane,
  - $\Omega$ : longitude of the ascending node, i.e. the angle measured in the equatorial plane between the ascending node of the orbit and the x-axis. The ascending node is defined to be the intersection between the Earth's equatorial plane and the satellite's orbit as it goes from the southern hemisphere to the northern one,
  - $\omega$ : argument of perigee, i.e. the angle between the perigee and the ascending node, measured in the orbital plane, and
  - $M$ : the mean anomaly, i.e. the time that has passed since the last passage at perigee, as a function of the orbital period, and expressed as an angle. It is one of the three anomalies including the eccentric anomaly  $E$  and the true anomaly  $\theta$  which can be used instead. They are related by Kepler's equation  $M = E - e \cdot \sin E$  and by the relation  $\cos \theta = \frac{\cos E - e}{1 - e \cdot \cos E}$ .

In a Keplerian motion with no perturbation, the mean anomaly is the only orbital elements that varies with respect to time:

$$M = M_0 + n(t - T_0).$$

## 2.4 Coordinate Transformations

**Orbital Elements**  $\longrightarrow$  **PQW-coordinates:**

Initial conditions given in the form of orbital elements  $(a, e, i, \Omega, \omega, M)$  can be converted to Cartesian variables in the PQW-frame using a simple coordinate

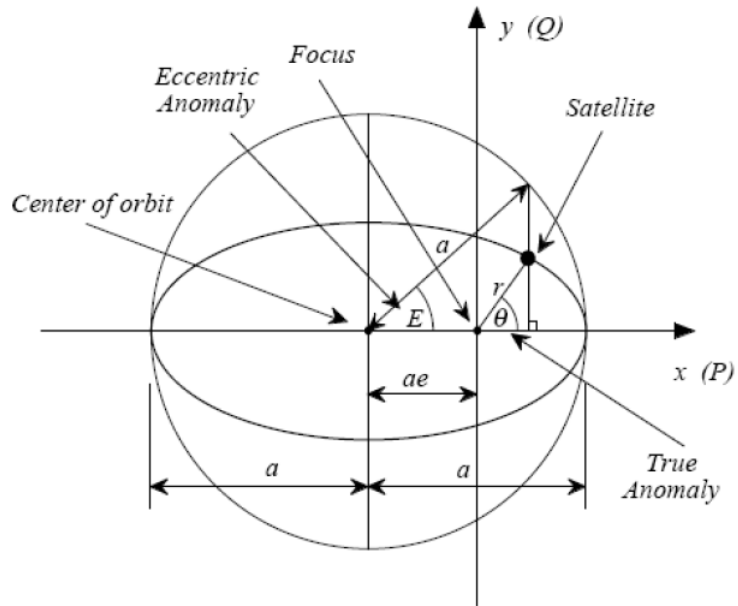


Figure 2.3: Satellite Orbital Elements ( $a$ ,  $e$ ,  $E$ ,  $\theta$ )

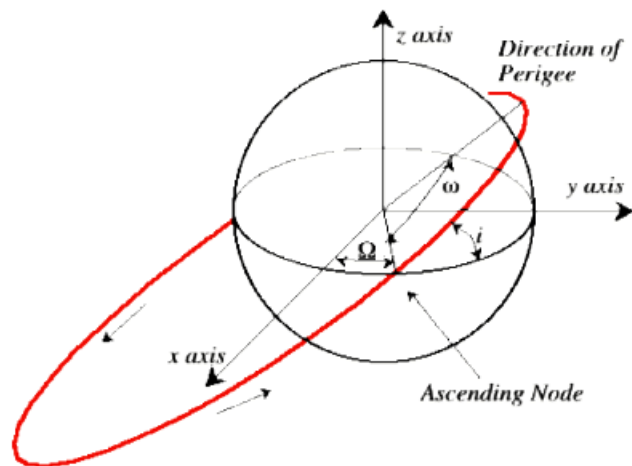


Figure 2.4: Satellite Orbital Elements ( $i$ ,  $\omega$ ,  $\Omega$ )

transformation given in [34]. The position and velocity vectors are thus:

$$\vec{r}_{PQW} = \begin{pmatrix} a(\cos E - e) \\ a\sqrt{1-e^2} \sin E \\ 0 \end{pmatrix} \quad \text{and} \quad \vec{r}'_{PQW} = \begin{pmatrix} -\frac{na \sin E}{1-e \cos E} \\ \frac{na\sqrt{1-e^2} \cos E}{1-e \cos E} \\ 0 \end{pmatrix} \quad (2.12)$$

**PQW-coordinates  $\longleftrightarrow$  ECI-coordinates:**

The following three rotations are needed to go from the ECI-frame to the PQW-frame:

- $r_Z(\Omega)$ : rotate by  $\Omega$  about the Z-axis,
- $r_X(i)$ : rotate by  $i$  about the new X-axis,
- $r_Z(\omega)$ : rotate by  $\omega$  about the new Z-axis.

These rotations are characterized by three rotation matrices, namely,  $R_Z(\Omega)$ ,  $R_X(i)$  and  $R_Z(\omega)$ , respectively.

The base vectors of the PQW-frame  $(\vec{P}, \vec{Q}, \vec{W})$  are given in terms of the base vectors of the ECI-frame  $(\vec{i}, \vec{j}, \vec{k})$ :

$$\begin{pmatrix} \vec{P} \\ \vec{Q} \\ \vec{W} \end{pmatrix} = A \begin{pmatrix} \vec{i} \\ \vec{j} \\ \vec{k} \end{pmatrix} \quad (2.13)$$

where  $A$  is the transformation matrix, whose entries are given in terms of the orbital elements:

$$A = \begin{pmatrix} \cos \omega \cdot \cos \Omega - \sin \omega \cdot \cos i \cdot \sin \Omega & \cos \omega \cdot \sin \Omega + \sin \omega \cdot \cos i \cdot \cos \Omega & \sin \omega \cdot \sin i \\ -\sin \omega \cdot \cos \Omega - \cos \omega \cdot \cos i \cdot \sin \Omega & -\sin \omega \cdot \sin \Omega + \cos \omega \cdot \cos i \cdot \cos \Omega & \sin i \cdot \cos \omega \\ \sin i \cdot \sin \Omega & -\sin i \cdot \cos \Omega & \cos i \end{pmatrix}. \quad (2.14)$$

Note that the transformation matrix  $A$  is orthogonal ( $A^{-1} = A^T$ ).

Therefore one can go from the cartesian coordinates  $(x_1, y_1, z_1)$  in the ECI-frame to the cartesian coordinates  $(x_2, y_2, z_2)$  in the PQW-frame by using

the following relation:

$$\begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}_{PQW} = A \begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}_{ECI} \quad (2.15)$$

Alternatively, one can go from the cartesian coordinates  $(x_2, y_2, z_2)$  in the PQW-frame to the cartesian coordinates  $(x_1, y_1, z_1)$  in the ECI-frame using:

$$\begin{pmatrix} x_1 \\ y_1 \\ z_1 \end{pmatrix}_{ECI} = A^T \begin{pmatrix} x_2 \\ y_2 \\ z_2 \end{pmatrix}_{PQW} \quad (2.16)$$

**Orbital Elements  $\rightarrow$  ECI-coordinates:**

Using (2.12) and (2.13), one gets the expressions of the position and velocity vectors in the ECI-frame given the orbital elements:

$$\begin{cases} \vec{r}_{ECI} = a(\cos E - e) \vec{P}_{ECI} + a\sqrt{1-e^2} \sin E \vec{Q}_{ECI} \\ \dot{\vec{r}}_{ECI} = -\frac{na \sin E}{1-e \cos E} \vec{P}_{ECI} + \frac{na\sqrt{1-e^2} \cos E}{1-e \cos E} \vec{Q}_{ECI} \end{cases} \quad (2.17)$$

where  $\vec{P}_{ECI}$ ,  $\vec{Q}_{ECI}$  and  $\vec{W}_{ECI}$  are found using (2.13).

**ECI-coordinates  $\rightarrow$  Orbital Elements**

Given the position and velocity vectors:  $\vec{r}_{ECI} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} r_1 \\ r_2 \\ r_3 \end{pmatrix}$  and

$$\dot{\vec{r}}_{ECI} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} \dot{r}_1 \\ \dot{r}_2 \\ \dot{r}_3 \end{pmatrix}$$

The semi-major axis  $a$  is computed using:

$$\frac{1}{a} = \frac{2}{r} - \frac{v^2}{\mu} \quad (2.18)$$



Where:  $r^2 = x^2 + y^2 + z^2$  and  $v^2 = \dot{x}^2 + \dot{y}^2 + \dot{z}^2$

The eccentricity  $e$  and the eccentric anomaly  $E$  are computed using the following system of equations:

$$\begin{cases} e \cos E = t_1 = \frac{rv^2}{\mu} - 1 \\ e \sin E = t_2 = \frac{\|\vec{r}\| \|\dot{\vec{r}}\|}{\sqrt{\mu a}} \end{cases} \quad (2.19)$$

Therefore:

$$\begin{cases} e = \sqrt{t_1^2 + t_2^2} \\ E = \tan^{-1} \frac{t_2}{t_1} \end{cases} \quad (2.20)$$

The mean anomaly  $M$  can also be computed:  $M = E - e \sin E = E - t_2$ . Consequently, the above calculated orbital elements are used in order to find for  $i = 1, 2, 3$ :

$$\begin{cases} P_i = \frac{r_i}{r} \cos E - \dot{r}_i \sqrt{\frac{a}{\mu}} \sin E \\ Q_i = \frac{\frac{r_i}{r} \sin E + \dot{r}_i \sqrt{\frac{a}{\mu}} (\cos E - e)}{\sqrt{1 - e^2}} \end{cases}$$

At this stage, the inclination  $i$ , the argument of the perigee  $\omega$  and the longitude of the ascending node  $\Omega$  are calculated:

$$\begin{cases} i = \tan^{-1} \frac{\sqrt{P_3^2 + Q_3^2}}{P_1 Q_2 - P_2 Q_1} \\ \omega = \tan^{-1} \frac{P_3}{Q_3} \\ \Omega = \tan^{-1} \frac{P_2 Q_3 - P_3 Q_2}{P_1 Q_3 - P_3 Q_1} \end{cases} \quad (2.21)$$

## 2.5 $J_2$ -perturbed Motion as a System of ODE's

### 2.5.1 Expression of $\vec{\nabla}U$ in the ECI-frame

**Explicit Expression of  $U$ :**

$$\text{Let } \vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \dot{\vec{r}} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \text{ and } \ddot{\vec{r}} = \begin{pmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{pmatrix}.$$

The gravitational potential is given by (2.7):

$$U = u(J_2) = -\frac{\mu}{r} \left\{ 1 + \left( \frac{r_{eq}}{r} \right)^2 J_2 P_2(\sin \varphi) \right\}.$$

where:  $r = \sqrt{x^2 + y^2 + z^2}$ .

First of all, we note that this potential  $U = u(J_2)$  is the sum of a “Keplerian term”, which is the term corresponding to central component of the Earth’s gravitational force field, and a second term which is due to the  $J_2$ -perturbation:

$$U = U_K + U_P, \tag{2.22}$$

where:

$$U_K = -\frac{\mu}{r}, \tag{2.23}$$

and:

$$U_P = -\frac{\mu}{r} \left( \frac{r_{eq}}{r} \right)^2 J_2 P_2(\sin \varphi). \tag{2.24}$$

Recall that  $\mu$ ,  $r_{eq}$  and  $J_2$  are constants given by (2.3), (2.8) and (2.9) respectively.

In order to get an expression of  $U_P$  in terms of  $(x, y, z)$ , it is necessary to express the geocentric latitude  $\varphi$  in terms of these coordinates.  $\varphi$  is the angle that the line connecting the center of the Earth and the satellite, makes with the Earth’s equatorial plane. Note that the north latitude is positive, while the south latitude is negative.

And since the  $xy$ -plane of the ECI frame is in fact the equatorial plane, it follows that  $\sin \varphi$  is expressed in the ECI-frame as follows:

$$\sin \varphi = \left[ \frac{z}{r} \right]_{ECI},$$

and thus:

$$P_2(\sin \varphi) = \frac{3}{2} \sin^2 \varphi - \frac{1}{2} = \left[ \frac{3z^2}{2r^2} - \frac{1}{2} \right]_{ECI}.$$

Therefore:

$$U_P = - \left[ \frac{3\mu J_2 r_{eq}^2 z^2}{2r^5} - \frac{\mu J_2 r_{eq}^2}{2r^3} \right]_{ECI}. \quad (2.25)$$

### Expression of $\vec{\nabla}U$ :

Note that:

$$\vec{\nabla}U = \vec{\nabla}U_K + \vec{\nabla}U_P,$$

and let:

$$\vec{f}_K(\vec{r}) = -\vec{\nabla}U_K$$

and:

$$\vec{f}_P(\vec{r}) = -\vec{\nabla}U_P,$$

therefore:

$$-\vec{\nabla}U = \vec{f}_K(\vec{r}) + \vec{f}_P(\vec{r}), \quad (2.26)$$

In a coordinate system where  $\vec{r} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$  and  $r = \sqrt{x^2 + y^2 + z^2}$ , one can express  $-\vec{\nabla}U_K$  as:

$$\vec{f}_K(\vec{r}) = \vec{f}_K \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} \frac{\partial U_K}{\partial x} \\ \frac{\partial U_K}{\partial y} \\ \frac{\partial U_K}{\partial z} \end{pmatrix} = \begin{pmatrix} -\mu \frac{x}{r^3} \\ -\mu \frac{y}{r^3} \\ -\mu \frac{z}{r^3} \end{pmatrix}. \quad (2.27)$$

And one can obtain  $-\vec{\nabla}U_P$  in the ECI-frame, where the explicit expression for  $U_P$  is given by (2.25):

$$\left[ \vec{f}_P \begin{pmatrix} x \\ y \\ z \end{pmatrix} \right]_{ECI} = \begin{pmatrix} \frac{\partial U_P}{\partial x} \\ \frac{\partial U_P}{\partial y} \\ \frac{\partial U_P}{\partial z} \end{pmatrix}_{ECI} = \begin{pmatrix} \frac{3\mu J_2 r_{eq}^2}{2} \left( 1 - 5 \frac{z^2}{r^2} \right) \frac{x}{r^5} \\ \frac{3\mu J_2 r_{eq}^2}{2} \left( 1 - 5 \frac{z^2}{r^2} \right) \frac{y}{r^5} \\ \frac{3\mu J_2 r_{eq}^2}{2} \left( 3 - 5 \frac{z^2}{r^2} \right) \frac{z}{r^5} \end{pmatrix}_{ECI} \quad (2.28)$$

## 2.5.2 Equivalent System of First Order ODE's

The differential equation (2.11) modelizing the  $J_2$  problem along with a set of initial conditions, yields a second-order initial value problem, in which one seeks  $\vec{r}$  where:

$$\begin{cases} \vec{r}(t) = \vec{f}(\vec{r}), & t > 0, & (2.29.1) \\ \vec{r}(0) = \vec{r}_0, & & (2.29.2) \\ \vec{r}'(0) = \vec{r}'_0, & & (2.29.3) \end{cases} \quad (2.29)$$

with:

$$\vec{f}(\vec{r}) = -\vec{\nabla}U. \quad (2.30)$$

### Lowering the Order of the Differential Equation:

Define:

$$\vec{r}_1(t) = \vec{r}(t) \quad \text{and} \quad \vec{r}_2(t) = \vec{r}'(t).$$

Thus solving the second order problem (2.29) would be equivalent to solving the following first order problem for  $\vec{r}_1, \vec{r}_2$ :

$$\begin{cases} \frac{d\vec{r}_1}{dt} = \vec{r}_2, & t > 0, & (2.31.1) \\ \frac{d\vec{r}_2}{dt} = \vec{f}(\vec{r}_1), & t > 0, & (2.31.2) \\ \vec{r}_1(0) = \vec{r}_0, & & (2.31.3) \\ \vec{r}_2(0) = \vec{r}'_0. & & (2.31.4) \end{cases} \quad (2.31)$$

Let:

$$Y = \begin{pmatrix} \vec{r}_1 \\ \vec{r}_2 \end{pmatrix} \in \mathbb{R}^6, \quad Y_0 = \begin{pmatrix} \vec{r}_1(0) \\ \vec{r}_2(0) \end{pmatrix} = \begin{pmatrix} \vec{r}_0 \\ \vec{r}'_0 \end{pmatrix} \in \mathbb{R}^6.$$

Thus one can rewrite problem (2.31) as a first-order initial value problem of dimension 6, of the general form (S), in which one seeks  $Y : [0, \infty) \rightarrow \mathbb{R}^6$  which satisfies:

$$\begin{cases} \frac{dY}{dt} = F(Y), & t > 0, \\ Y(0) = Y_0 \end{cases} \quad (2.32)$$

where:

$$F(Y) = \begin{pmatrix} \vec{r}_2 \\ \vec{f}(\vec{r}_1) \end{pmatrix} \in \mathbb{R}^6.$$

Since  $U = U_K + U_P$  (2.22), it implies that one can also split  $F(Y)$  into a sum of two terms, a Keplerian term and a perturbing term:

$$F(Y) = F_K(Y) + F_P(Y), \quad (2.33)$$

with:

$$F_K(Y) = \begin{pmatrix} \vec{r}_2 \\ \vec{f}_K(\vec{r}_1) \end{pmatrix}, \quad (2.34)$$

$$F_P(Y) = \begin{pmatrix} \vec{0} \\ \vec{f}_P(\vec{r}_1) \end{pmatrix}. \quad (2.35)$$

**Explicit Expression of  $F(Y)$ , in the ECI-frame:**

$$[\vec{r}]_{ECI} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} \text{ and } [\vec{r}']_{ECI} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix} \text{ yield: } [Y]_{ECI} = \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \\ Y_6 \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \\ \dot{x} \\ \dot{y} \\ \dot{z} \end{pmatrix}.$$

One deduces then, using (2.27) and (2.28) that:

$$[F(Y)]_{ECI} = [F_K(Y)]_{ECI} + [F_P(Y)]_{ECI},$$

where:

$$[F_K(Y)]_{ECI} = \begin{pmatrix} Y_4 \\ Y_5 \\ Y_6 \\ -\mu \frac{1}{R^3} \begin{pmatrix} Y_1 \\ Y_2 \\ Y_3 \end{pmatrix} \end{pmatrix} \quad (2.36)$$

and:

$$[F_P(Y)]_{ECI} = \frac{3\mu J_2 r_{eq}^2}{2} \begin{pmatrix} 0 \\ 0 \\ 0 \\ \frac{3\mu J_2 r_{eq}^2}{2} \begin{pmatrix} 1 - 5 \frac{Y_3^2}{R^2} \\ 1 - 5 \frac{Y_3^2}{R^2} \\ 3 - 5 \frac{Y_3^2}{R^2} \end{pmatrix} \begin{pmatrix} \frac{Y_1}{R^5} \\ \frac{Y_2}{R^5} \\ \frac{Y_3}{R^5} \end{pmatrix} \end{pmatrix} \quad (2.37)$$

with  $R = r = \sqrt{Y_1^2 + Y_2^2 + Y_3^2}$ .

Note that  $R = \|\vec{r}\|_2$  is the distance between the position of the satellite and the earth's center and therefore,  $R$  is independent of the coordinates system in which is expressed  $\vec{r} = (Y_1, Y_2, Y_3)$ .

# Chapter 3

## Adaptive Parallel Time

### Integration

#### 3.1 Sequential Solvers

If we consider the first order initial value problem:

$$(S) \quad \begin{cases} \frac{dY}{dt} = F(Y), & 0 < T_0 < t \leq T, \\ Y(T_0) = Y_0, \end{cases} \quad (3.1)$$

where one seeks to find the solution  $Y : [T_0, T] \rightarrow \mathbb{R}^k$ . Assuming that the existence and uniqueness of the solution is established on  $[0, \infty]$ , we aim in this thesis to solve Eq. 3.1 using parallel time methods. We start first by examining the traditional approaches to solve such IVPs. Numerical algorithms for IVPs discretize the time interval  $[T_0, T]$  by time steps of  $\tau$ , then proceed by one-step or multi-step discretization which lead to a discrete set of points  $\{Y_i | i = 0 \dots n\}$  at time  $\{T_i = T_0 + i\tau | i = 0 \dots n\}$  with  $T_n = T$ . For instance, the Euler explicit algorithm proceeds as shown in Algorithm 1.

---

**Algorithm 1** Euler Explicit

---

**Input:**  $Y_0, T_0, T_{max}, F$   
 $counter = 1;$   
 $Y(counter) = Y_0;$   
 $t(counter) = T_0;$   
**while**  $t(counter) < T_{max}$  **do**  
     $K1 = \tau \times F(Y(counter));$   
     $counter = counter + 1;$   
     $Y(counter) = Y(counter - 1) + K1;$   
     $t(counter) = t(counter - 1) + \tau;$   
**end while**

---

Note that this method has first order convergence where:

$$\max_{1 \leq i \leq n} |Y(T_i) - Y_i| = O(\tau)$$

On the other hand, the most popular method used to solve Eq. 3.1 is the fourth order Runge-Kutta which follows the procedure shown in Algorithm 2.

---

**Algorithm 2** Runge-Kutta of order 4

---

**Input:**  $Y_0, T_0, T_{max}, F$   
 $counter = 1;$   
 $Y(counter) = Y_0;$   
 $t(counter) = T_0;$   
**while**  $t(counter) < T_{max}$  **do**  
     $temp = Y(counter);$   
     $K1 = \tau \times F(temp);$   
     $K2 = \tau \times F(temp + \frac{1}{2}K1);$   
     $K3 = \tau \times F(temp + \frac{1}{2}K2);$   
     $K4 = \tau \times F(temp + K3);$   
     $temp = temp + \frac{1}{6} \times (K1 + 2 \times K2 + 2 \times K3 + K4);$   
     $counter = counter + 1;$   
     $Y(counter) = temp;$   
     $t(counter) = t(counter - 1) + \tau;$   
**end while**

---



This Runge-Kutta method has fourth order convergence rate, where:

$$\max_{1 \leq i \leq n} |Y(T_i) - Y_i| = O(\tau^4)$$

There are also implicit schemes such as the Euler-implicit scheme which solves a non-linear system at every time step, namely

$$\begin{cases} Y_{i+1} - \tau F(Y_{i+1}) = Y_i \\ Y(0) = Y_0 \end{cases} \quad (3.2)$$

## 3.2 Parallel Solvers

In the context of integrating an ordinary differential equation, there is no natural parallelism one can use. However in recent years, many attempts have been made to try and take advantage of parallel computer architectures in order to speed up the computations required in such a problem. Even though the different methods differ substantially but they can be organized in three categories as shown in [6], [18] and [1]:

- **Parallelism across the method:**

These methods attempt to split the computations to be performed in one integration step, among many parallel computing units. This approach requires the redesign of the original sequential code in order to be more suiting to the parallel architecture used. At the basic level, these methods can make use of concurrent function evaluations within a single integration step, for example in the case of Runge-Kutta methods. This route may prove itself useful when the function evaluations are costly. These methods may on the other hand employ computing blocks of values simultaneously which can be done for instance by block predictor-corrector methods. But this route is more efficient when using a few powerful processors because of communication constraints.

- **Parallelism across the system:** These methods rely on the decomposition of the problem, to be solved, into smaller sub-problems which

can be solved in parallel at that stage. One of the simplest temporal iterative approaches is the Picard method according to [5]. Consider a problem of size  $k$ , this method generates a sequence of iterative solutions in the region of integration which allows for a very natural parallelism by decoupling the problem into  $k$  independent problems.

- **Parallelism across time(across the steps):** These methods attempt to perform many integration steps simultaneously. This approach requires the redesign of the original sequential code in order to be more suiting to the parallel architecture used. In fact, time parallelism has not received much attention in the literature because of the inherent sequential nature of the time integration process. Hence space parallelism has always been more popular than time parallelism according to [12]. But when space parallelism is not possible, time parallelism becomes of great importance especially when:

- The number of spatial degrees of freedom in the problem is small.
- The solution’s time advancing needs to be in real time.
- The problem cannot take advantage of all the available processors.

In this thesis, we are concerned with parallelism across time.

### 3.2.1 Time Parallelism

Time parallelism is somewhat counter-intuitive since traditional sequential solvers start from the initial value  $Y_0$  at time  $T_0$  and use it in order to advance to another value  $Y_1$  at time  $T_1 = T_0 + \tau$  using a certain procedure. Afterwards the resulting value  $Y_1$  at time  $T_1$  is used in order to get to value  $Y_2$  at time  $T_2 = T_1 + \tau$ . And this procedure continues until the final time in the interval  $T$  is reached. Hence, computations cannot be carried out in the third time slice for instance unless the results for the first and second slices are calculated.

The novelty here is to employ multiple shooting approaches for the purpose of time parallelism. These approaches are based on the concept of time domain decomposition and have been investigated thoroughly in [29], [7], [23],[9] and [12]. As a first step the time interval  $[T_0, T]$  is discretized into  $N$  time slices, which constitutes a coarse grid. Then the solution is predicted at these coarse grid points, and these predictions are used as initial conditions

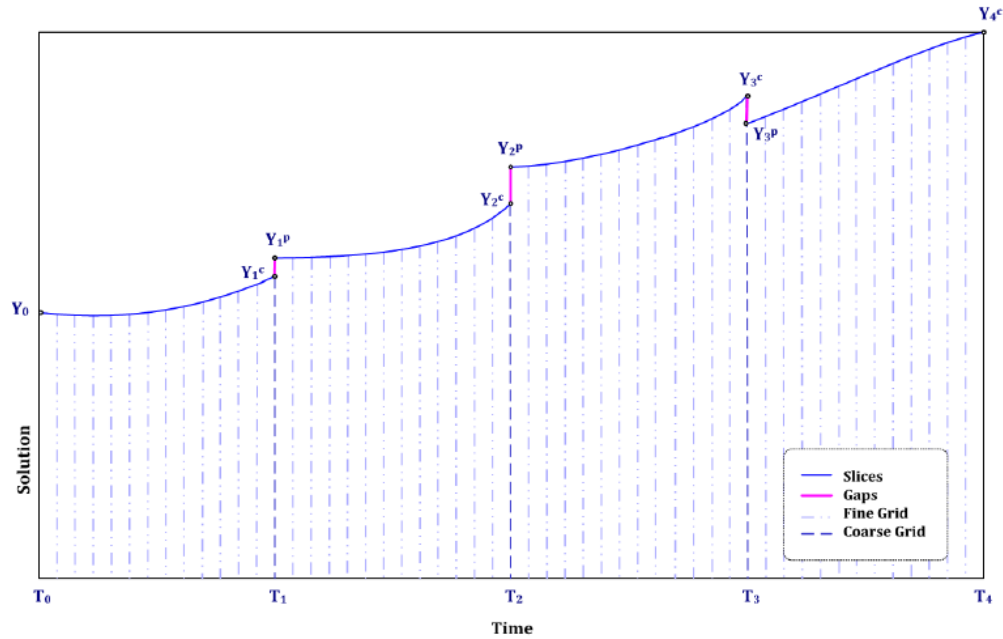


Figure 3.1: Multiple shooting method

for each independent slice hence turning the original problem into  $N$  independent problems. Subsequently, these  $N$  problems can be solved in parallel starting at the initial value at each time slice in order to get to a final value at the end of this time slice. This brings about discrepancies or "gaps" between the predicted values and the computed values at the coarse grid points as shown in Fig. 3.1. At that stage, a corrective step is carried through to improve the prediction seeds and minimize the gaps until continuity is achieved. At that point global convergence of the algorithm is attained.

### General Steps of Multiple-Shooting Methods

1. Coarse grid discretization: The interval of integration  $[T_0, T]$  is decomposed into  $N$  time slices, resulting into a set of time values  $[T_0 \dots T_N]$  which constitutes the coarse time grid.
2. Initial Prediction: Initial prediction of the values of the solution  $\{Y_n^p\}$  at every time grid point.

3. Iterative process: Do until convergence of all slices:

- *Parallel integration*: on each of the time slices  $[T_{n-1}, T_n]$ , using a fine grid to solve the independent initial value sub-problems:

$$\begin{cases} \frac{dY}{dt} = F(Y), & T_{n-1} < t \leq T_n, \\ Y(T_{n-1}) = Y_{n-1}^p. \end{cases}$$

This process leads to a computed value of the solution  $Y_n^c$  at the end of each time slice  $[T_{n-1}, T_n]$ .

- *Check for convergence*: on every time slice by evaluating the "gaps" between the predicted values  $\{Y_n^p\}$  and the computed values  $\{Y_n^c\}$  at every coarse time grid point.

**If** the gaps are acceptable within a certain tolerance, then the iterative process stops and convergence is achieved.

**Else** the iterations continue.

- *Corrective process*: executed in order to update the predicted values at the onset of each time slice.

One of these time-parallel approaches used to solve time-dependent problems is the "**A**daptive **P**arallel **T**ime **I**ntegration" (**APT**I) that was devised in [27] and [22].

### 3.3 Adaptive Parallel Time Integration

APT I is a predictor-corrector scheme that has been devised in [27] and [21]. It is based on a sliced-time approach found in [26] and was successfully implemented to unbounded solutions appearing in reaction diffusion problems and oscillating membranes in [33], [20] and [28].

The novelty in this approach is brought about by:

- The automatic generation of time-slices by using an **E**nd **O**f **S**lice(EOS) condition to terminate the time slice.

- Detecting a ratio property.
- Rescaling the problem.
- Making ratio-based predictions and corrections.

### 3.3.1 Automatic Generation of Time-Slices: End Of Slice

#### (EOS) Condition

As in all previous time parallel approaches, the APTI procedure divides the time interval  $[T_0, T]$  into sub-intervals or time slices  $[T_{i-1}, T_i]$  such that:

$$\bigcup_{i=1 \dots N} [T_{i-1}, T_i] = [T_0, T] \quad (3.3)$$

But these time slices are not uniform but are rather generated automatically by the APTI algorithm. It does so by using a stopping criterion in order to produce the coarse grid, made up of slices which satisfy 3.3. We shall call this criterion the EOS condition, which is of the form:

$$E(Y(t)) = 0 \quad (3.4)$$

Therefore, the coarse grid is generated such that the end of slice values  $\{Y_i | i = 1, \dots, N\}$  satisfy Equation 3.4. The coarse grid starts off by containing only  $\{T_0\}$  and then it is built incrementally from that point on. As a first step, that is on the 1<sup>st</sup> slice starting from the initial value of  $Y_0$  at time  $T_0$ , we seek  $\{Y_1, T_1\}$  using the following procedure:

- Start solving the system:

$$\begin{cases} \frac{dY}{dt} = F(Y), & T_0 < t \leq T_1, \\ Y(T_0) = Y_0, \end{cases} \quad (3.5)$$

- Keep stepping forward in time until a value of  $Y(t)$  is reached such that the EOS condition  $E(Y(t)) = 0$  is reached for the first time.

At that point we record that time value  $T_1$  as a coarse time grid value. Thus the coarse grid comprises now of  $\{T_0, T_1\}$ .

In general, on the  $i^{\text{th}}$  slice starting from the initial value of  $Y_{i-1}$  at time  $T_{i-1}$ , we seek  $\{Y_i, T_i\}$  using the following procedure:

- Start solving the system:

$$\begin{cases} \frac{dY}{dt} = F(Y), & T_{i-1} < t, \\ Y(T_{i-1}) = Y_{i-1}, \end{cases} \quad (3.6)$$

- Keep stepping forward in time until a value of  $Y(t)$  is reached such that the EOS condition is reached for the first time since being satisfied at  $T_{i-1}$ .

At that point we record that time value  $T_i$  as a coarse time grid value and update the coarse grid to  $\{T_0, T_1, \dots, T_i\}$ . And so on for the rest of the slices until we reach  $T_N$ .

Therefore we get end of slice values  $(T_i, Y_i = Y(T_i))$  such that for all  $0 < i \leq N$ :

$$\begin{cases} \text{At } t = T_i, & E(Y(T_i)) = 0, \\ \forall t \in ]T_{i-1}, T_i[, & E(Y(t)) \neq 0. \end{cases} \quad (3.7)$$

### 3.3.2 Selection of an EOS condition

The selection of the function  $\{E\}$  should be made such that the End of Slice condition 3.4 is satisfied infinitely many times, hence yielding a unique coarse grid. Therefore, this selection is in fact problem-dependant and takes into consideration the global behaviour of the solution.

Thus far two cases of existence of a function  $\{E\}$  have been established according to [22]:

- **Oscillating Solutions**

This case is applicable when the general behaviour of the solution  $\{Y\}$  is oscillatory, over a long period of time. That is when there exists a two-dimensional plane  $P$  in  $\mathbb{R}^k$  on which the projection of  $\{Y\}$  rotates about a fixed center  $q$ . Therefore the EOS condition is chosen to be at the instance when the solution  $\{Y\}$  executes a full rotation about  $q$  in the plane  $P$ .

For instance, let  $Y = [xy]^t$ . Consider the trivial periodic problem:

$$\begin{cases} \frac{dY}{dt} = Y^p \\ Y(T_0) = Y_0 \end{cases} \quad (3.8)$$

where  $Y > 0$  and  $p \in \mathbb{R}$ .

Here the solution  $\{Y\}$  is periodic in the phase plane. Hence a good choice would be to end each of the slices when the polar angle  $\theta$  returns to its initial value  $\theta_0$  modulo  $2\pi$ . Thus the EOS condition would be:

$$\text{atan2}(y_i, x_i) - \text{atan2}(y_0, x_0) = 0 \quad (3.9)$$

That produces a coarse grid of equal time slices with size  $\delta T$  equal to the period of the solution  $\{Y\}$ , and equal EOS values of  $Y$ :  $\frac{Y_i}{Y_{i-1}} = 1$ .

- **Explosive Solutions**

This case applies to problems when  $\lim_{t \rightarrow \infty} \|Y\|_{\infty} = 0$ . Here we need to choose the stopping condition in a way to prevent the relative growth of  $Y_i$  with respect to  $Y_{i-1}$  from surpassing a certain threshold  $S$ . Hence, we choose to end each slice at time  $T_i$  such that:

$$\|Y_i ./ Y_{i-1}\|_{\infty} = 1 + S \quad (3.10)$$

Note that the operator “./” denotes the component-wise division.

In the case where  $Y$  is a scalar, the EOS condition is formulated as:

$$\frac{Y_i}{Y_{i-1}} - (1 + S) = 0 \quad (3.11)$$

### Equivalent Initial Value Shooting Problems

The enforcement of a coarse grid on the time interval transforms the Initial Value Problem 3.1 into an equivalent sequence of Initial Value Shooting Problems where one seeks on the  $i^{\text{th}}$  slice  $[T_{i-1}, T_i]$  the EOS solution value  $Y_i$  and the EOS time  $T_i$  such that:

$$(S_i) \quad \begin{cases} \frac{dY}{dt} = F(Y), & T_{i-1} < t \leq T_i \\ Y(T_{i-1}) = Y_{i-1}, \\ E(Y(T_i)) = 0, & \text{and } \forall t \in ]T_{i-1}, T_i[, E(Y(t)) \neq 0 \end{cases} \quad (3.12)$$

### 3.3.3 Ratio vectors and Ratio property

On the  $i^{\text{th}}$  slice  $[T_{i-1}, T_i]$ , we define the ratio vector  $R_i$  to be the ratio of the end of slice value of the solution to the starting value at the beginning of this slice, namely:

$$R_i = \begin{pmatrix} R_{i,1} \\ \vdots \\ R_{i,j} \\ \vdots \\ R_{i,k} \end{pmatrix} \quad (3.13)$$

Where the components of the vector are defined as:

$$R_{i,j} = \frac{Y_{i,j}}{Y_{i-1,j}} \quad \forall 1 \leq j \leq k \quad (3.14)$$

We assume without loss of generality that the solution sequence  $\{Y_i\}$



satisfies the following condition:

$$\forall i, \forall j : 1 \leq j \leq k, Y_{i,j} \neq 0 \quad (3.15)$$

Where  $Y_{i,j}$  is the  $j^{th}$  component of the solution vector  $Y_i$  at time  $T_i$ .  
Therefore, we deduce that:

$$\forall i \geq 1, Y_i = R_i \cdot * Y_{i-1} \quad (3.16)$$

Where “ $\cdot*$ ” is the component wise multiplication. Thus we can formulate the following recurrence relation:

$$Y_i = R_i \cdot * R_{i-1} \cdot * \dots * R_1 \cdot * Y_0 \quad (3.17)$$

Thus the initial prediction of the values  $\{Y_i\}$  can be obtained from the the prediction of the ratio values  $\{R_i\}$ , which is possible when the EOS values exhibit a ratio property as defined below.

## Ratio Property

The solution values  $\{Y_i\}$  of the initial value shooting problems 3.12 exhibit a *ratio property* if:

$$\exists i_0, \forall i > i_0, \|R_{i+1} - R_i\|_\infty < \epsilon \quad (3.18)$$

Where  $\epsilon$  is some tolerance.

There are many categories of the ratio property.

- **Perfect Ratio Property:** A sequence of solution values  $\{Y_i\}$  are said

to have this property if there exists a ratio vector  $R_1$  such that:

$$\forall i > 0, R_i = R_1.$$

- **Asymptotic Ratio Property:** A sequence of solution values  $\{Y_i\}$  are said to have this property if the sequence of ratio vectors  $\{R_i\}$  converge to a certain vector  $R_L$ :

$$\lim_{i \rightarrow \infty} R_i = R_L$$

- **Weak Ratio Property:** A sequence of solution values  $\{Y_i\}$  are said to have this property if the sequence of ratio vectors  $\{R_i\}$  satisfy the following condition on  $n_p$  consecutive slices:

$$\forall i \in \{i_0 + 1, \dots, i_0 + n_p\}, \|R_i - R_{i-1}\|_\infty < \epsilon$$

When the sequence of values exhibit a ratio property then one can make good ratio-based predictions as we shall later see. This property is very much dependant on the choice of an EOS condition.

### 3.3.4 Change of variables

At the core of the APTI method lies a rescaling methodology which allows the performance of parallel computations, even when the initial time of a time slice is not known in advance.

On every time slice, the solution  $Y$  and the time variable  $t$  are changed into a rescaled solution  $Z_i$  and a rescaled time  $s$  respectively, using the following transformation:

$$\begin{cases} t = T_{i-1} + \beta_i s, & \beta_i > 0 \\ Y(t) = Y_{i-1} + D_i Z_i(s), \end{cases} \quad (3.19)$$

Where:

- $\beta_i$  is a time rescaling factor which is chosen in such a way that the solution can be controlled. And the computation on the rescaled system is therefore equivalent or "similar", that is they are solved using the same numerical solver with identical rescaled time steps. It is only at the end of the slices that the rescaled time steps are adjusted in order to reach the EOS condition. Note that  $\beta_i$  modulates the fixed rescaled time steps(in terms of  $s$ ) to produce adaptive real time steps(in terms of  $t$ ).
- $Y_{i-1} = Y(T_{i-1}) \in \mathbb{R}^K$
- $D_i = \text{diag}(\alpha_i) \in \mathbb{R}^{K \times K}$  is a diagonal matrix where the diagonal entries are the elements in the vector  $\alpha_i \in \mathbb{R}^K$ , which are defined as follows:

$$\alpha_i[j] = \begin{cases} Y_{i-1}[j], & \text{if } Y_{i-1}[j] \neq 0 \\ 1, & \text{if } Y_{i-1}[j] = 0 \end{cases} \quad (3.20)$$

In fact the diagonal entries  $\alpha_i[j]$  are defined in Eq. 3.20 to insure that the matrix  $D_i$  is invertible. Hence on every time slice, we have the rescaled solution vector:

$$Z_i(s) = D_i^{-1}(Y(t) - Y_{i-1}) \quad (3.21)$$

That is for  $j \in \{1, 2, \dots, K\}$ :

$$Z_i(s)[j] = \begin{cases} \frac{Y_i(t) - Y_{i-1}[j]}{Y_{i-1}[j]}, & \text{if } Y_{i-1}[j] \neq 0 \\ Y_i(t), & \text{if } Y_{i-1}[j] = 0 \end{cases} \quad (3.22)$$

And the rescaled time is given by  $s = \frac{t - T_{i-1}}{\beta_i}$ .

Note that in the rescaled systems, the rescaled solution  $Z_i(s)$  and the rescaled time  $s$  are both set to 0 at the beginning of every slice. Therefore every initial value shooting problem ( $S_i$ ) can be solved locally as shown in Fig. 3.2.

Note also that the end of slice rescaled time which corresponds to  $T_i$  is:  $s_i = \frac{T_i - T_{i-1}}{\beta_i}$ . Therefore  $s_i$  is the size of the  $i^{\text{th}}$  rescaled slice, since the rescaled time  $s = 0$  at the beginning of the slice.

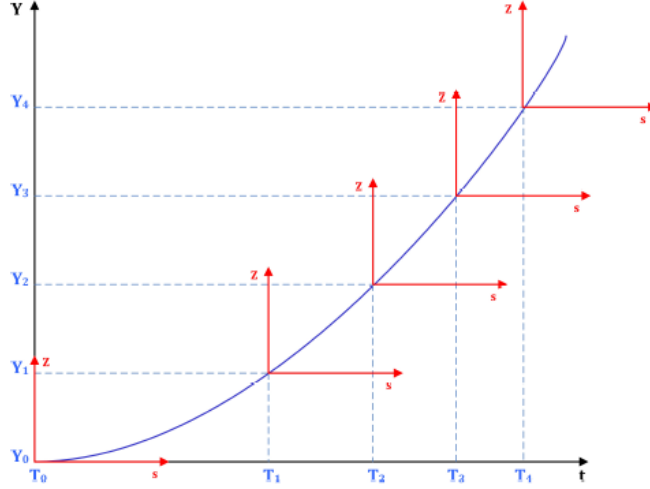


Figure 3.2: Solving rescaled systems locally

### End of slice invariances:

The solution function  $Z_i$  surely depends on the choice of  $\beta_i$ , but the following end of slice invariances are valid independently of  $\beta_i$ . From the change of variables 3.19, we have at the end of the  $i^{\text{th}}$  slice:

- $$\forall \beta_i, T_i = T_{i-1} + \beta_i s_i, \quad (3.23)$$

therefore:

$$\forall \beta_i, \Delta T_i = T_i - T_{i-1} = \beta_i s_i. \quad (3.24)$$

That is, the product  $\beta_i s_i$  is independent of the choice of  $\beta_i$ , and is equal to the size of the  $i^{\text{th}}$  slice  $\Delta T_i$ .

- $$\forall \beta_i, Y_i = Y_{i-1} + D_i Z_i(s_i), \quad (3.25)$$

therefore:

$$\forall \beta_i, Z_i(s_i) = D_i^{-1}(Y_i - Y_{i-1}). \quad (3.26)$$

That is, rescaled solution end of slice values  $Z_i(s_i)$  are independent of  $\beta_i$ .

These end of slice invariance identities are at the core of our prediction model that will be discussed later on in this chapter, where if the choice of  $\beta_i$  are such that the behavior of  $\{s_i, Z_i(s_i)\}$  can be accurately predicted, then the pair  $\{T_i, Y_i\}$  can be obtained from 3.23 and 3.25 respectively.

### Case of Nonzeroness:

In general the choice of the EOS condition is made in order to satisfy the following non-zeroness condition:

$$\forall i, \forall j : 1 \leq i \leq K, Y_i[j] \neq 0. \quad (3.27)$$

Therefore  $\alpha_i = Y_{i-1}$ , and thus the change of variables is:

$$\begin{cases} t = T_{i-1} + \beta_i s, & \beta_i > 0 \\ Y_i = D_i(\mathbf{1} + Z_i(s_i)), \end{cases} \quad (3.28)$$

Where  $D_i$  is a diagonal matrix having  $Y_{i-1}$  on its diagonal, and  $\mathbf{1}$  is a vector of ones of dimension  $K$ . Therefore:

$$Z_i(s_i) = D_i^{-1}Y_i - \mathbf{1} \quad (3.29)$$

Note that in this case  $Z_i(s_i)$  can be expressed in terms of the ratio vector  $R_i = D_i^{-1}Y_i = Y_n./Y_{n-1}$ . So the behavior of  $Z_i(s_i)$  is equivalent to that of  $R_i$ .

Also note that the general choice of  $\alpha_i$  3.20 allows to tackle problems where there are some zero components in the solution vector  $\{Y_i\}$ .

### Resulting Rescaled Systems

The original Initial Value Problem (S) is equivalent to a sequence of rescaled Initial Value Shooting Problems where one seeks the rescaled solution function  $Z_i(s_i)$  and the end of slice rescaled time  $s_i$  on the slice  $[T_{i-1}, T_i]$ , in the

form:

$$(S'_i) \quad \begin{cases} \frac{dZ_i}{ds} = G_i(Z_i), & 0 < t \leq s_i \\ Z_i(0) = 0, \\ H(Z_i(s_i)) = 0, \text{ and } \forall s < s_i, H_i(Z_i(s)) \neq 0 \end{cases} \quad (3.30)$$

where

$$G_i(Z_i) = \beta_i D_i^{-1} F(Y_{i-1} + D_i Z_i)$$

Note that the functions  $G_i$  depends on the starting values  $T_{i-1}$  and  $Y_{i-1}$ .

## Properties of the Rescaled Initial Value Shooting Problems:

All the rescaled initial value shooting problems start at an initial solution value of zero and end at the same EOS condition. These problems have some similarity properties:

- **Invariance:**

The rescaled problems (3.30) are invariant if the EOS functions  $\{H\}$  are invariant and the functions  $\{G_i\}$  are also invariant given the choice of a critical rescaling parameter  $\{\beta_i\}$ , that is:

$$\forall n, \quad G_i(\cdot) = G_1(\cdot) \quad (3.31)$$

In this case all the rescaled Initial Value Problems ( $S'_i$ ) will be in the form:

$$\begin{cases} \frac{dZ_1}{ds} = G_1(Z_1), & 0 < s \leq s_1 \\ Z_1(0) = 0, \\ H(Z_1(s_1)) = 0 \quad \forall s < s_1, H(Z_1(s)) \neq 0 \end{cases} \quad (3.32)$$

Thus all the rescaled systems will have the same solution  $\{s_1, Z_1(s_1)\}$ .

This is the ideal case of similarity according to [21]. One needs to solve the problem on only one time slice and get the solution on all time-slices through a change of variables. But note that rescaling does not guarantee invariance.

An interesting case of invariance is the problem of calculating a satellite's orbit in a Keplerian motion.

- **Asymptotic Similarity:**

The systems  $(S'_n)$  are asymptotically similar if the rescaling parameters  $\{\beta_i\}$  are chosen such that the functions  $\{G_i\}$  converge uniformly to a certain function  $G_L$ , as well as if the rescaled EOS functions  $\{H\}$  are invariant:

$$\lim_{i \rightarrow \infty} \|G_i(W) - G_L(W)\|_{\infty} = 0 \quad (3.33)$$

Where  $G_L$  defines a limit shooting value problem:

$$\begin{cases} \frac{dZ_L}{ds} = G_L(Z_L), & 0 < t \leq s_L \\ Z_L(0) = 0, \\ H(Z_L(s_L)) = 0, \text{ and } \forall s < s_L, H(Z_L(s)) \neq 0 \end{cases} \quad (3.34)$$

Whose solution pair  $\{s_L, Z_L(s_L)\}$  is such that:  $\lim_{i \rightarrow \infty} \{s_i, Z_i(s_i)\} = \{s_L, Z_L(s_L)\}$ .

In this case we can use rescaling 3.2 for prediction purposes after running a sequential solver on a certain number of slices  $n_s$ . At that point we have:

$$\max_{i > n_s} \{ \max\{|s_i - s_{i-1}|, \|Z_i(s_i) - Z_{i-1}(s_{i-1})\|\} \} \leq tol \quad (3.35)$$

where  $tol$  is a tolerance set by the user.

- **Numerical Similarity:**

This case can be used when there is no proof of invariance or asymptotic similarity.

The rescaled systems 3.30 present numerical "weak" similarity on  $n_r$  consecutive slices starting at slice  $n_0$  if the EOS functions  $\{H\}$  are

invariant and if :

$$\max_{n_0 \leq i \leq n_0 + n_r} \{ \max\{|s_i - s_{i-1}|, \|Z_i(s_i) - Z_{i-1}(s_{i-1})\|\} \} \leq tol \quad (3.36)$$

In this case we let  $n_s = n_0 + n_r$ .

This case of weak similarity is encountered in the perturbed motion of a satellite.

### 3.3.5 Ratio-Based Prediction and Correction Procedures

#### Ratio-Based Prediction

We can deduce from the recurrence relation 3.17 that the prediction of the ratio vectors  $R_i$  can be utilized for the prediction of the solution  $Y_i$  at  $T_i$  in terms of any previous EOS solution value  $Y_l$  such that  $l < i$ , by using the following formula:

$$Y_i[j] = Y_l[j] * (R_{l+1}[j]) * (R_{l+2}[j]) \dots * (R_i[j]) \quad (3.37)$$

We might encounter two cases for the ratio vectors, perfect and non-perfect.

- **Perfect Ratio Property:**

This is the case when the ratio vectors are invariant, that is

$$\forall i, \quad R_i = R_0.$$

In this case the sequential computations need to be done on a single slice in order to make exact predications:

$$\forall i, \forall j, \quad Y_i[j] = Y_0[j] * R_0[j].$$



- **Non-Perfect Ratio Property:**

This is the case when the ratio vectors are not invariant, hence the prediction process requires us to run a preliminary sequential procedure on  $n_s$  slices, where  $n_s$  is small relative to the number of slices  $N$ .

If the ratios stabilize when we reach the  $n_s$  slice, up to a desired tolerance, then the last ratio  $R_{n_s}$  is used in order to predict the subsequent  $Y_i$ :

$$Y_i^p[j] = Y_{n_s}[j] * R_{n_s}^{i-n_s}[j], \quad \forall i > n_s, \forall j. \quad (3.38)$$

Where  $Y^p$  denotes the predicted value of the solution  $Y$ .

Otherwise if the ratios do not stabilize by the  $n_s$  slice, then we need to run a backward analysis on the exact ratios calculated on the first  $n_s$  slices and then use a mathematical model, namely extrapolation, that approximates the sequence  $\{R_i\}$ , ( $i \leq n_s$ ). This model is then used in order to predict ratios  $\{R_i^p\}$  ( $i > n_s$ ). Once that is done we can predict the next initial solution values  $\{Y_i^p\}$  using the recurrence relation 3.17:

$$Y_i^p[j] = Y_{n_s}[j] * (R_{n_s+1}^p[j]) * (R_{n_s+2}^p[j]) * \dots * (R_i^p[j]). \quad (3.39)$$

## Ratio-Based Correction Procedure

The correction procedure is similar to the prediction procedure, but with first updating  $n_s$  with the number of the converged slices at the previous iteration. Then the same process discussed above is employed. In the next section, we introduce the APTI algorithm in details.

## 3.4 APTI Algorithm

At the basis on any parallel in time algorithm, lies the following relation:

$$\max \left\{ \frac{\|Y_{i-1} - Y_{i-1}^p\|}{\|Y_{i-1}\|}, \frac{|T_{i-1} - T_{i-1}^p|}{|T_{i-1}|} \right\} = O(tol) \Rightarrow \max \left\{ \frac{\|Y_i - Y_i^c\|}{\|Y_i\|}, \frac{|T_i - T_i^c|}{|T_i|} \right\} = O(tol) \quad (3.40)$$

**Theorem 1.** *Assuming 3.40 is satisfied, then:*

$$\left\{ \begin{array}{l} \max \left\{ \frac{\|Y_{i-1} - Y_{i-1}^p\|}{\|Y_{i-1}\|}, \frac{|T_{i-1} - T_{i-1}^p|}{|T_{i-1}|} \right\} = O(tol) \\ \max \left\{ \frac{\|Y_i^p - Y_i^c\|}{\|Y_i^p\|}, \frac{|T_i^p - T_i^c|}{|T_i^p|} \right\} = O(tol) \end{array} \right\}$$

$$\Rightarrow \left\{ \begin{array}{l} \max \left\{ \frac{\|Y_i - Y_i^p\|}{\|Y_i\|} \right\} \\ \max \left\{ \frac{|T_i - T_i^p|}{|T_i|} \right\} \end{array} \right\} = O(tol)$$

Therefore an iterative procedure can now be initiated using a parallel architecture with P processors.

The Adaptive Parallel Time Integration procedure features the following steps according to [21]:

- **Step 1: Preliminary Sequential Run**

Every processor solves the first  $n_s$  slices of the rescaled shooting value problems and computes the successive ratio vectors  $\{R_i\} = \{Y_i./Y_{i-1}\}$ , and keeps going until the detection of a ratio property, that is when the ratios stabilize up to a certain tolerance  $\epsilon_R$ .

- **Step 2: Iterative Process**

- **2.1: Predict**

On the slices  $i > n_s$ , every processor fits the last few ratios into an appropriate mathematical model then extrapolates in order to compute predicted solution values  $Y_i^p$ .

- **2.2: Parallel Run on the Remaining Slices**

The remaining slices, that is for slices numbered  $i$ , where  $i > n_s$  are assigned to the processors available based on a cyclic distribution. For instance, if there are two available processors, one takes on the computations on the odd numbered slices and the other takes the even numbered slices. If one has 4 available processors, the first processor will be assigned slices numbered  $n$  such that  $n \bmod 4$  is equal to 1. The second processor will be as shown in Fig.

3.3. And so on... Given  $n_p$  processors, the  $k^{th}$  processor will be assigned slices:  $k, k + n_p, k + 2n_p, \dots$  and will solve the  $i^{th}$  slice starting with the predicted solution value  $Y_{i-1}^p$  to get a calculated value  $Y_i^c$ , with a computational tolerance  $\epsilon_{tol}$ .

– **2.3:** Calculate the Gaps

Each processor reaches the end of every slice by calculating  $Y_i^c$ . It now calculates the difference between  $Y_i^c$  and the value  $Y_i^p$  that was predicted in Step (2.1). It now calculates the sequence of relative gaps at the end of the  $i^{th}$  slice, for  $i > n_s$  as follows:

$$G_i = \frac{Y_i^c - Y_i^p}{\max(|Y_i^c|, |Y_i^p|)} \quad (3.41)$$

Every processor now finds  $\|G_i\|_\infty$ , and checks if  $\|G_i\|_\infty \leq \epsilon_G$  where  $\epsilon_G$  is some chosen tolerance on the relative gaps, then the processor proceeds to solve slice numbered  $i + n_p$ . That is every processor checks if the relative difference between the predicted value  $Y^p$  and the calculated value  $Y^c$  at the end of every slice, it continues on to solve the next slice assigned to it until it has solved all slices assigned to it. Note that this step is done in parallel on all processors but with every processor solving different slices. See Fig. 3.4.

– **2.4:** Broadcast  $n_{last}$  and  $Y_{n_{last}}$

If the relative gap  $\|G_i\|_\infty$  was larger than  $\epsilon_G$ , this indicates that the predicted values  $Y_i^p$  were not sufficiently accurate. Therefore we need to do a correction on these predicted values. In order to do that, every processor should send the number of the last few slices that converged  $n_{last}$  and the values corresponding to these slices  $Y_{n_{last}}$ , to all other processors.

– **2.5:** Update  $n_s$  Each processor now:

- \* Calculates the number of the last slice that has converged  $n_{conv} = \max_{1 \leq j \leq n_p} \{n_{last}\}_j$ , i.e.  $n_{conv}$  is the maximum of  $n_{last}$  values collected from all processors.
- \* Updates  $n_s$  by  $n_{conv}$ .
- \* Repeats Step (2) until convergence of all slices.

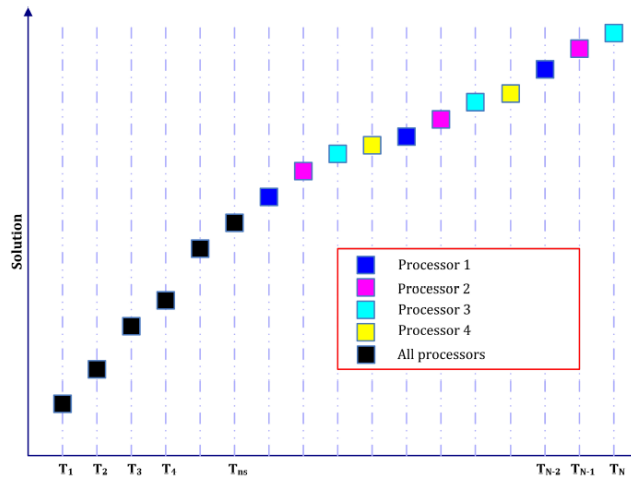


Figure 3.3: Cyclic Distribution of Slices Among 4 Processors

- **Step 3: Broadcast Time Vector**

When all the slices have converged, every processor then broadcasts the sizes of its time slices according to [20] to all other processors. This allows the calculation of the global time vector. This step is necessary since as we mentioned before time is set to zero at the beginning of every rescaled time slice, in order to allow the processor to start solving a slice even though the starting time of that slice is not known in advance since the previous slice is assigned to another processor.

When Step (3) is complete, every processor will have the global solution  $Y(t)$ .

## 3.5 Increase Speed-Up through a Duplication

### Approach

A duplication approach lies in executing sequential procedures simultaneously on all available processors within a parallel algorithm. The alternative "classical" approach would be to let one processor execute these sequential

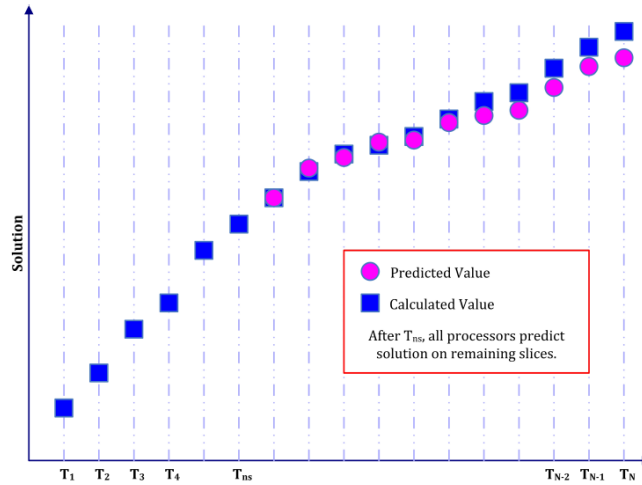


Figure 3.4: Gaps Between Calculated and Predicted Values of the Solution

procedures and then communicate the results to all remaining processors. But this approach of course involves a communication cost that is not present in the duplication approach since all the processors would do the same work at the same time, hence at the end they will all have the results needed without needing to communicate among them, as shown in Fig 3.5. Using the duplication approach would of course decrease the communication overhead according to [21] and hence reduce the time of parallel execution of the algorithm, thus increasing speed-up and efficiency. Additionally, one would avoid idle time on any processor.

**Note** that in the APTI algorithm, the sequential parts of the algorithm are Steps (1: Preliminary Sequential Run) and (2.1: Prediction).

In the "classical" implementation of the APTI algorithm displayed in Fig 3.5, a single processor finds the predicted values  $Y_i^p$  then sends them to all other processors. Then every processor solves in parallel the slices assigned to it in order to find the computed values  $Y_i^c$ . Afterwards all the processors send the computed values to the master processor and wait until it computes the new predictions and sends these predictions back to all other processors. This process is repeated until convergence of all slices.

While in the duplication approach, the sequential predictions are done by all the processors at the same time hence one needs only 1 communication step in each iteration rather than 2.

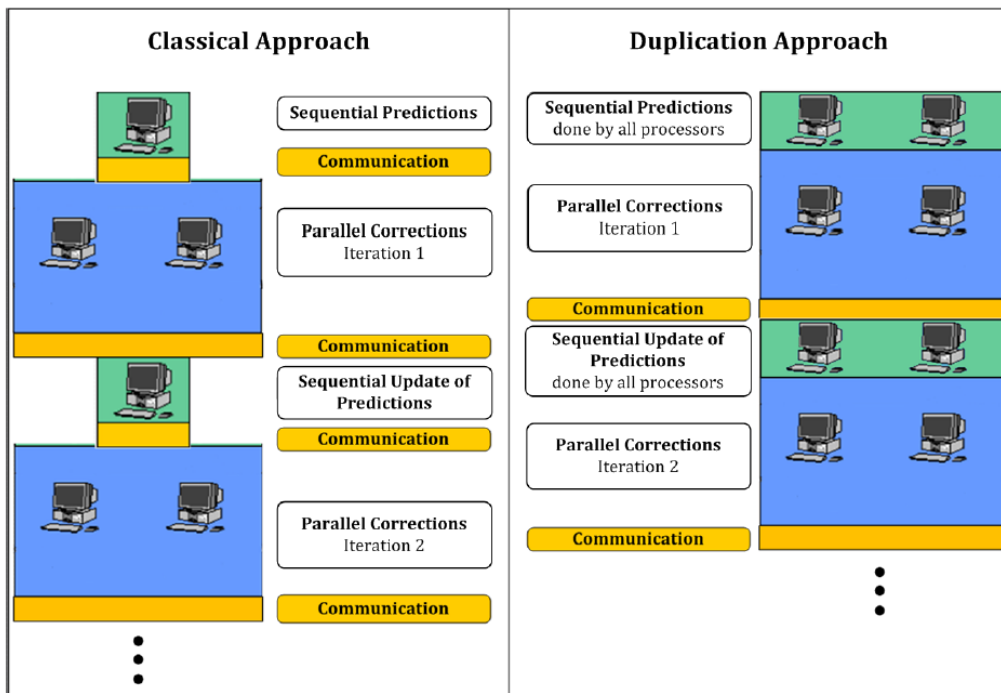


Figure 3.5: Classical versus Duplication Approach for APTI

# Chapter 4

## The Parareal Algorithm: Parallel in Real Time

### 4.1 Related Work

In 1964, Nievergelt proposed the first parallel time integration method in [29]. This method led to the multiple shooting method which solve systems of differential equations of the form:

$$u' = f(u), \quad u(0) = u_0, \quad t \in [0, T] \quad (4.1)$$

The first step of these methods is in dividing the time interval  $[0, T]$  into  $N$  subintervals  $0 = T_0 < T_1 < \dots < T_{N-1} < T_N = T$  of size  $\delta T = \frac{T}{N}$  and thus ending up with  $N$  independent Initial Value Problems:

$$\begin{cases} u'_n = f(u_n), & \forall n = 0, 1, \dots, N, \\ u_n(T_{n-1}) = U_{n-1}, & \text{where } U_n = u(T_n) \end{cases} \quad (4.2)$$

Then continuity conditions at the end of the subintervals are applied:

$$\begin{cases} U_0 - U_0 = 0, \\ U_1 - u_1(T_1) = 0, \\ \vdots \\ U_{N-1} - u_{N-1}(T_{N-1}) = 0, \\ U_N - u_N(T_N = T) = 0. \end{cases} \quad (4.3)$$

This takes the form of a system of nonlinear equations:

$$F(U) = 0, \quad U = (U_0, U_1, \dots, U_N)^T \quad (4.4)$$

Chartier and Philippe in [8], as well as Erhel and Rault [9], used an iterative Newton procedure to solve 4.4. This procedure involves solving:

$$J_F(U^k)[U^{k+1} - U^k] = -F(U^k), \quad k = 0, 1, \dots \quad (4.5)$$

Where  $J_F$  is the Jacobian of the function  $F$  defined in 4.3. This procedure is stopped when it reaches a given accuracy.

Rearranging Eq 4.5 gives the following update equation:

$$U^{k+1} = U^k - J_F^{-1}(U^k)F(U^k) \quad (4.6)$$

Exploiting the structure of Eq 4.3, one can rewrite the update term  $J_F^{-1}(U^k)F(U^k)$  in the form:

$$\begin{bmatrix} I & & & & & \\ -\frac{\partial u_0}{\partial U_0}(t_1, U_0^k) & I & & & & \\ & -\frac{\partial u_1}{\partial U_1}(t_2, U_1^k) & I & & & \\ & & \ddots & \ddots & & \\ & & & -\frac{\partial u_{N-1}}{\partial U_{N-1}}(t_{n-1}, U_{N-1}^k) & I & \end{bmatrix}^{-1} \begin{pmatrix} U_0^k - u_0 \\ U_1^k - u_1(t_1, U_0^k) \\ U_2^k - u_2(t_2, U_1^k) \\ \vdots \\ U_N^k - u_{N-1}(T, U_{N-1}^k) \end{pmatrix} \quad (4.7)$$

4.6 thus produces the following recurrence relation of the multiple shoot-



ing method applied to an initial value problem:

$$\begin{cases} U_0^{k+1} = u_0, \\ U_{n+1}^{k+1} = u_n(t_{n+1}, U_n^k) + \frac{\partial u_n}{\partial U_n}(t_{n+1}, U_n^k)(U_n^{k+1} - U_n^k) \end{cases} \quad (4.8)$$

At iteration  $k$ , computations on the independent slices yield end values  $\{u_n(t_{n+1}, U_n^k)\}$ , then one uses the recurrence 4.8 to correct these values in order to end up with  $U_{n+1}^{k+1}$ , which are then used as initial values for iteration  $k + 1$ .

In order to implement this method, one needs a numerical method to compute  $\{u_n\}$  and a method to compute or approximate the terms  $\frac{\partial u_n}{\partial U_n}(t_{n+1}, U_n^k)(U_n^{k+1} - U_n^k)$  of the Jacobian.

In 2001, a new predictor-corrector scheme was introduced by Lions, Maday and Turinici [23], the "Parareal algorithm". There have been many developments made to this algorithm. In [11] Farhat and Chanderesis offered a mathematical justification for the Parareal algorithm based on the theory of distributions. In [16] Gander and Vandewalle prove that the parareal algorithm is a multiple shooting method where one approximates the Jacobian by a finite difference approach. In [11] Farhat et al presented the application of this algorithm to fluid and structure applications.

## 4.2 LIONS Parareal Algorithm

Lions Parareal algorithm was a turning point in the time parallel solution of time-dependent differential equations. We find that many authors have tried to apply this algorithm on a variety of problems from different application areas. Gander et al provided a convergence analysis of the Parareal algorithm in [16], [14]. He also provided a derivation of this algorithm in [14]. We decided to apply this algorithm to the Satellite problem and test its fast convergence.

Both the Parareal algorithm and the APTI algorithms are predictor corrector schemes as mentioned in the prior sections. They are both iterative procedures, in the sense that they consist of iterating until convergence is reached. And in every iteration, prediction and correction is done.

The Parareal algorithm is defined using two propagation operators:

- $F(t_2, t_1, u_1)$  provides an accurate approximation of the solution  $u(t_2)$  given the initial condition  $u(t_1) = u_1$ .
- $G(t_2, t_1, u_1)$  provides a less accurate approximation of the solution  $u(t_2)$  given the initial condition  $u(t_1) = u_1$ , for example on a coarser grid or using a lower order method, or even an approximation using a simpler model.

The algorithm starts with the initial condition  $U_0^0 = u_0$ . It then obtains an initial approximation of  $U_n^0, n = 0, \dots, N$  at times  $t_0, t_1, \dots, t_N$  using the sequential computation of  $U_{n+1}^0 = G(t_{n+1}, t_n, U_n^0)$ . It then performs for  $k = 0, 1, \dots$  the correction iteration:

$$U_{n+1}^{k+1} = G(t_{n+1}, t_n, U_n^{k+1}) + F(t_{n+1}, t_n, U_n^k) - G(t_{n+1}, t_n, U_n^k) \quad (4.9)$$

Note that the Parareal algorithm 4.9 will for  $k \rightarrow \infty$  converge to a series  $U_n$  which satisfies  $U_{n+1} = F(t_{n+1}, t_n, U_n)$ . That is the approximation of the solution values  $U_n$  will converge to the values obtained using the fine sequential propagator F.

The parareal method was first introduced in [23] and applied to a linear scalar model. In [2] the new simplified form 4.9 was introduced by Baffico et al. In what follows is the derivation presented by Gander and Vanderwalle in [15] based on the multiple shooting method introduced in the previous subsection, applied to 4.1.

The formulation 4.8 is continuous, but in order to apply the multiple shooting algorithm it is necessary to discretize the differential equations for every sub- time interval. One should select a numerical method in order to compute  $u_n(t_{n+1}, U_n^k)$ , and one should chose a method to compute or approximate the terms  $\frac{u_n}{U_n}(t_{n+1}, U_n^k)$  or their effect on the difference term  $U_n^{k+1} - U_n^k$ .

If we approximate in the multiple shooting relation 4.8 the terms:

1.  $u_n(t_{n+1}, U_n^k)$  by the solution given by the fine propagator  $F(t_{n+1}, t_n, U_n^k)$
2.  $\frac{\partial u_n}{\partial U_n}(t_{n+1}, U_n^k)(U_n^{k+1} - U_n^k)$  in a finite difference way using the G propagator  $G(t_{n+1}, t_n, U_n^{k+1}) - G(t_{n+1}, t_n, U_n^k)$

Then in that case the multiple shooting method is exactly the Parareal algorithm.

---

**Algorithm 3** Parareal Algorithm

---

```

 $U_0^0 = u_0$ 
//Iteration 0
//Initial prediction
for  $n = 0$  to  $N - 1$  do
     $\hat{G}_{n+1}^0 = G(t_{n+1}, t_n, U_n^0)$ 
     $U_{n+1}^0 = \hat{G}_{n+1}^0$ 
end for
//Parareal Iterations
for  $k = 0$  to  $K_{max}$  do
     $U_0^{k+1} = u_0$ 
    //Parallel Step
    for  $n = 0$  to  $N - 1$  do
         $\hat{F}_{n+1}^k = F(t_{n+1}, t_n, U_n^k)$ 
    end for
    //Sequential Step
    for  $n = 0$  to  $N - 1$  do
        // Predict
         $\hat{G}_{n+1}^{k+1} = G(t_{n+1}, t_n, U_n^{k+1})$ 
        //Correct
         $U_{n+1}^{k+1} = \hat{G}_{n+1}^{k+1} + \hat{F}_{n+1}^k - \hat{G}_{n+1}^k$ 
    end for
    //Check for convergence
    if  $|U_{n+1}^{k+1} - U_{n+1}^k| < \epsilon \forall n$  then
        BREAK
    end if
end for

```

---

## 4.3 Implementation

In our implementation of the Parareal algorithm, we divided the time interval  $[0, T]$  into  $N$  coarse intervals each of size  $\delta T = \frac{T}{N}$ . We chose the two propagation operators in the following manner:

- The G operator is given by the classical fourth order Runge Kutta method with coarse time step  $\delta T = \frac{T}{N}$ .
- The F operator is given by the classical fourth order Runge Kutta method with fine time step  $\tau = \frac{\delta T}{N_f} = \frac{T}{N*N_f}$

In the following, we present an example of a system of Ordinary Differential Equations on which we apply the Parareal algorithm and show results obtained.

### Example: Brusselator

The Brusselator is a system of ODEs that models diffusion in a chemical reaction, and is presented by:

$$\begin{aligned} \dot{x} &= A + x^2y - (B + 1)x, \\ \dot{y} &= Bx - x^2y. \end{aligned} \tag{4.10}$$

Where:

- $A = 1$  and  $B = 3$ .
- The initial conditions are  $x(0) = 0$ ,  $y(0) = 0$ .
- The time interval is given by  $[0, 12]$ .
- The number of coarse slices is given by:  $N = 32$ , which implies that the coarse time step  $\delta T = 0.375$  seconds.
- The number of fine slices is given by:  $N_f = 20$ , which implies that the fine time step  $\tau = 0.01875$  seconds.

Figure 4.1 shows the initial guess of the solution given by the coarse solver. The Parareal algorithm converges after 6 iterations to the solution shown in Figure 4.2 . In both figures 4.1 and 4.2, the green line represents

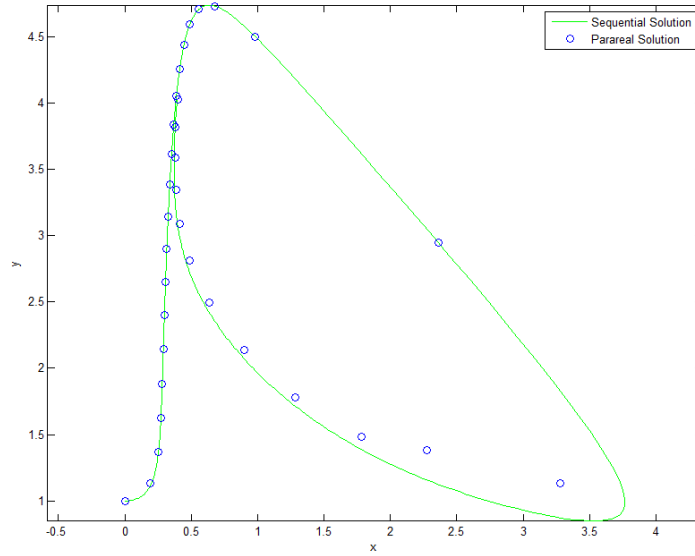


Figure 4.1: Initial guess of the solution of the Brusselator problem

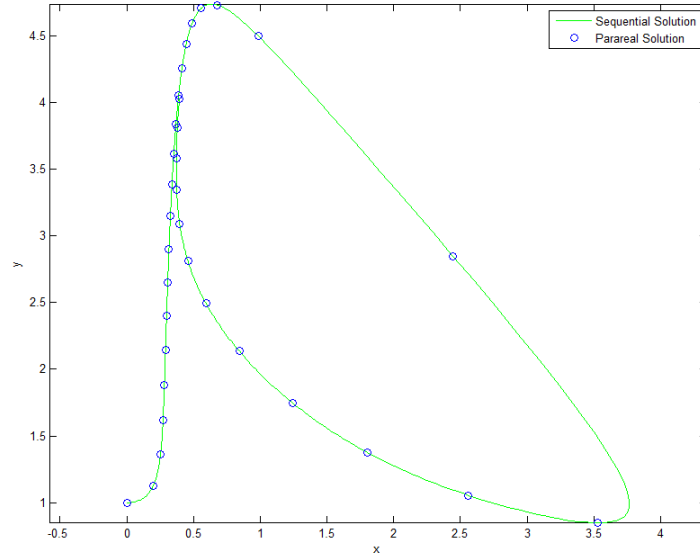


Figure 4.2: Final approximation of the solution of the Brusselator problem

the Sequential fine grid solution, while the blue circles represent the solution given by the Parareal algorithm. The error of the final Parareal solution versus the sequential solution is given by  $6.32e - 10$ .

## 4.4 APTI vs Parareal

APTI and Parareal are both predictor-corrector methods. But if one examines both algorithms closely, one notices that they differ completely in the way predictions and corrections are made. We list the differences between the two approaches in the following points:

1. The Parareal algorithm makes predictions by using the coarse operator  $G$  which involves using forth order Runge Kutta with a relatively ‘big’ time step  $\frac{T}{N}$ , while the APTI algorithm uses ratio-based predictions.
2. The initial predictions made by the Parareal algorithm involve only the sequential computation of the  $N$  slices using a coarse grid, while the predictions made by the APTI algorithm involve the sequential computation of  $n_s$  slices using a fine grid until the ratio stabilizes.
3. The predictions made by the Parareal algorithm are less accurate than those made by the APTI since the Parareal uses an operator  $G$  of low accuracy, while APTI uses the exact solution of the first  $n_s$  slices in order to make the predictions on later slices.
4. The predictions made by the Parareal algorithm are expensive since they must be done sequentially. This is due to the fact that the prediction of the value on a slice depends on the values on the previous slices, which obliges every processor to compute the predictions on all the slices even if he is only assigned a few number of slices in the fine propagation. While the predictions of APTI can be done in parallel since the prediction of the solution value on a slice is independent of the values on the previous slices. Hence every processor makes a smaller number of predictions.
5. In order to make corrections, the Parareal algorithm computes the jumps, i.e. the differences between the values predicted by the coarse propagator and the values computed using the fine propagator. It then resorts to the propagation of these jumps in order to compute the new corrected values. This propagation cannot be done except sequentially since the computation of the correction on any slice depends on the correction on all the previous slices. Therefore every processor requires all the values computed in any iteration in order to compute the new corrected values, which are the predicted values for the next iteration

in case there was no convergence. While the APTI algorithm uses ratio-based corrections which can be done in parallel, since at any iteration each processor only needs the converged vector from the previous iteration to compute the new corrected values.

6. In the Parareal algorithm, in any given iteration all the processors have to solve all the slices assigned to them before proceeding to the next iteration, even if none of the slices converge. This is due to the fact that the algorithm needs all the computed values for every time slice since all these values are needed in order to make the propagation of jumps. But we note that one propagation operation is relatively fast and inexpensive. While in the APTI algorithm, every processors proceeds solving the slices assigned to it, while these slices are converging withing a certain accuracy. And if a slice does not converge, then the processor stops the computations at that point and switches to the next iteration.

**Remark.** It is important to note that in most of the papers which apply Parareal to various problems, the author concentrates on the total number of iterations it takes for the algorithm to converge, rather than on the actual execution time. Therefore the author claims fast convergence of the Parareal algorithm when the number of iterations is small compared to the total number of slices  $N$ . But mostly there is no actual time measurement to show speed-ups versus the sequential integrator.

For instance, in [14] Gander and Hairer solve the Brusselator system of ODEs using the Parareal algorithm and mention that there is speed-up with respect to the sequential integrator if we neglect the cost of the coarse solver in the parareal algorithm. This is in fact not practical because even though one coarse step is not inexpensive but when done on  $N$  slices twice during every parareal iteration, then this cost can no longer be neglected.

Another note is that the authors are measuring speed-up as  $\frac{N}{\text{number of iterations}}$  assuming that they run the algorithm on  $N$  processors. Therefore for this numbers to be meaningful, one must have as many processors as there are coarse time steps. And the number of coarse steps will logically increase as we increase the total time  $T$ , hence the parareal's performance is expected to decrease when using a bigger time interval.



## 4.5 Modified Parareal

In real applications, execution time is far more important than the number of iterations made by an algorithm before convergence. In fact the Parareal algorithm suffers from a performance issue in the statement made by the authors in the literature so far, for instance in [15].

In every parareal iteration, the parallel fine propagation step and the sequential coarse prediction and correction steps are performed on all the slices for  $n = 0$  to  $N - 1$ . Yet when performing experiments on the parareal algorithm, we noticed that at every iteration some number of slices converge before the last iteration. Hence, performing computations on the slices that have already converged to a specific tolerance, is a redundant task that is consuming execution time.

We suggest a modified version of the parareal algorithm 4 where computations at every iteration are done on the slices that have not converged yet. This version will decrease the execution time of the algorithm without changing the number of iterations the algorithm takes to converge. It is a fact that the number of iterations made till convergence will remain unchanged, but the number of operations (both parallel and sequential) being performed at every iteration will decrease significantly.

We applied both this modified parareal algorithm and the classic parareal algorithm to the Brusselator problem 4.10 and obtained the solutions shown in Figure 4.3. In fact the solution given by the modified parareal algorithm coincides with that given by the classic parareal algorithm. We show in table 4.1 the performance results of both algorithms on the Brusselator problem. Both algorithms take the same number of iterations to converge, which is in this case 6 iterations, but the execution time of the modified parareal is less than that of the classic parareal. In fact, there is a 32% improvement in execution time of the modified parareal algorithm versus the classic parareal algorithm.

---

**Algorithm 4** Modified Parareal Algorithm

---

```
 $U_0^0 = u_0$   
//Iteration 0  
//Initial prediction  
for  $n = 0$  to  $N - 1$  do  
     $\hat{\mathcal{G}}_{n+1}^0 = G(t_{n+1}, t_n, U_n^0)$   
     $U_{n+1}^0 = \hat{\mathcal{G}}_{n+1}^0$   
end for  
num_converged_slices = 0  
//Parareal Iterations  
for  $k = 0$  to  $K_{max}$  do  
     $U_0^{k+1} = u_0$   
    //Parallel Step  
    for  $n = \text{num\_converged\_slices}$  to  $N - 1$  do  
         $\hat{\mathcal{F}}_{n+1}^k = F(t_{n+1}, t_n, U_n^k)$   
    end for  
    //Sequential Step  
    for  $n = \text{num\_converged\_slices}$  to  $N - 1$  do  
        // Predict  
         $\hat{\mathcal{G}}_{n+1}^{k+1} = G(t_{n+1}, t_n, U_n^{k+1})$   
        //Correct  
         $U_{n+1}^{k+1} = \hat{\mathcal{G}}_{n+1}^{k+1} + \hat{\mathcal{F}}_{n+1}^k - \hat{\mathcal{G}}_{n+1}^k$   
    end for  
    //Update the number of converged slices  
    if  $|U_{n+1}^{k+1} - U_{n+1}^k| < \epsilon$  then  
        num_converged_slices += 1  
    end if  
    //Check for convergence  
    if num_converged_slices =  $N$  then  
        BREAK  
    end if  
end for
```

---

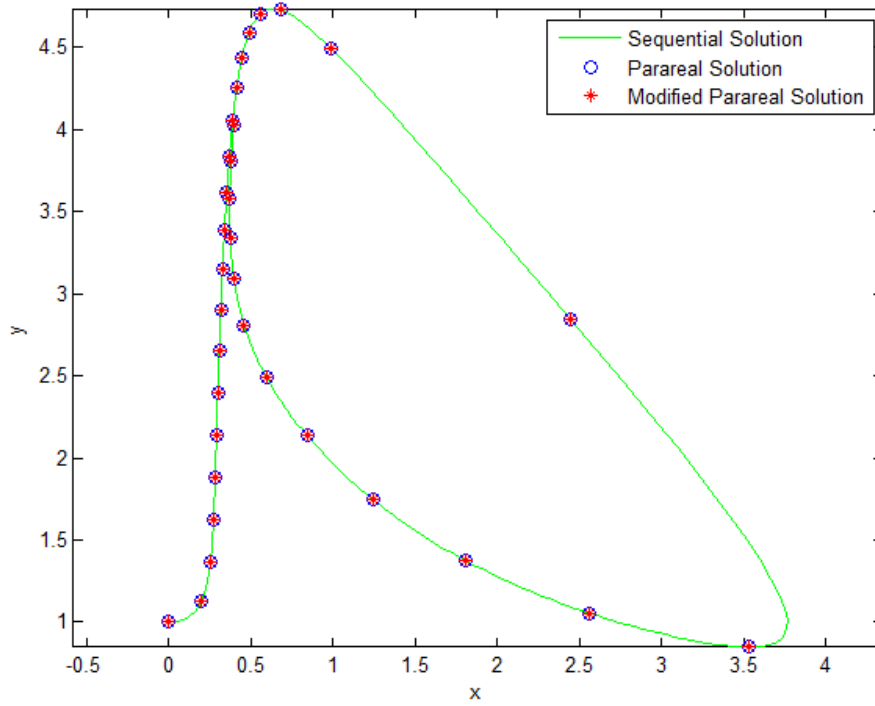


Figure 4.3: Solution of the Brusselator Problem given by the Parareal and Modified Parareal algorithms

Table 4.1: Performance results for the Brusselator problem

	Parareal	Modified Parareal
Number of Iterations	6	6
Execution time (seconds)	$2.9788 e - 01$	$2.0247 e - 01$

# Chapter 5

## Results

The following table 5.1 lists the main notations that will be used throughout this chapter:

Table 5.1: Notations used

$N$	Number of slices
$N_f$	Number of fine slices used by the fine propagator of the Parareal algorithm
$n_s$	Number of sequential slices computed by the APTI algorithm before the parallel computations start
$\tau$	Time step (in seconds) of integration used by the fine RK4 propagator
$T_s$	Time (in seconds) needed to solve the problem using a sequential procedure
$n_I$	Number of iterations it takes the algorithm to converge
$n_p$	Number of MATLAB workers used
$T_{n_p}$	Time (in seconds) needed to solve the problem in parallel using $n_p$ workers
$S_{n_p}$	Speed-up achieved while using $n_p$ workers
$E_{rel}$	Relative error of the parallel solution with respect to the sequential solution

The parallel speed-up  $S_{n_p}$  is evaluated as the ratio of the sequential exe-

cution time to the parallel execution time while using  $n_p$  MATLAB workers:

$$S_{n_p} = \frac{T_s}{T_{n_p}} \quad (5.1)$$

Moreover, the sequential execution is one where there is no predictions, corrections nor iterations. It is the applications of RK4 on the total number of slices.

The relative error is a measure of the difference between the solution given by the parallel solver  $Y^P$  and the solution given by the sequential solver  $Y^S$ :

$$E_{rel} = \frac{\|Y^P - Y^S\|}{\|Y^P\|} \quad (5.2)$$

Note that in MATLAB, one should preallocate vectors and matrices beforehand in both the sequential version and the parallel version of the algorithms. Increasing the size of vectors inside loops will adversely affect performance, since it implies repeatedly resizing arrays and looking for large continuous blocks of memory then moving the previous small array into these blocks. Not that in MATLAB, elements of a data structure are always stored in neighboring memory locations. Therefore, preallocating the maximum amount of memory space needed will contribute substantial improvement in execution time.

The tests were conducted using MATLAB R2013b.

## 5.1 Choice of an EOS condition for the APTI algorithm

The orbit of the satellite in a  $J_2$  perturbed motion, as stated in Chapter 2, is an osculating ellipse of which the orbital elements change at every instant. And all of these instantaneous ellipses have the center of the earth

at one focus, hence the satellite will certainly pass through the  $xy$ -plane of the IPQW frame, that is the plane corresponding to the Keplerian ellipse if there was no perturbation.

Moreover, we can consider the behavior of the solution to be oscillatory since there exists a two-dimensional plane (the  $xy$ -plane in the IPQW frame) on which the projection of  $\{Y\}$  rotates about a fixed center (the center of the earth). Therefore the EOS condition is chosen to be at the instance when the solution  $\{Y\}$  executes a full(or almost full) rotation about the center.

Therefore, a natural choice of the EOS condition would be to end the  $i^{th}$  slice when the satellite crosses the  $xy$ -plane, after completing a rotation, that is when:

$$Y_{i,3} = 0, \tag{5.3}$$

for the second time, after being satisfied at the previous slice  $Y_{i-1,3} = 0$ . This is due to the fact that having crossed the  $xy$ -plane at the end of the  $i - 1$  slice, the satellite will cross this plane twice in order to make an almost full rotation about the center of the earth, as shown in Figure 5.1. The behavior of the solution makes this EOS condition guaranteed to be reached.

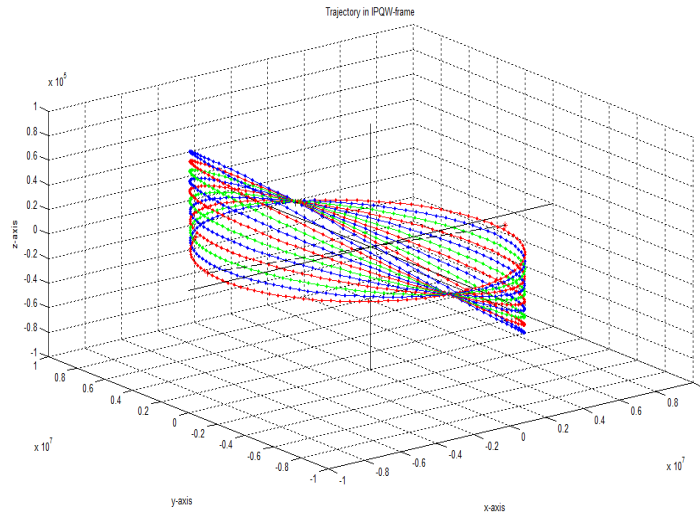


Figure 5.1: Orbit of a satellite in a  $J_2$  perturbed motion (11 rotations)

Let  $Q$  be the function defined by :

$$\begin{cases} Q[Y(t)] = 0, & \text{if } Y_3(t) = 0 \text{ for the second time after } Y_3(T_{i-1}) = 0 \\ Q[Y(t)] = 1, & \text{if } Y_3(t) \neq 0 \text{ or } Y_3(t) = 0 \text{ for the first time after } Y_3(T_{i-1}) = 0 \end{cases} \quad (5.4)$$

Thus the EOS condition 5.3 can be written in the following form:

$$Q[Y(T_i)] = 0 \quad \text{and} \quad \forall t \in ]T_{i-1}, T_i[, Q[Y(t)] \neq 0 \quad (5.5)$$

Hence the original IVP is equivalent to the sequence of initial value shooting problems where one seeks on the  $i^{\text{th}}$  slice  $[T_{i-1}, T_i]$  the EOS function  $Y_i$  and the EOS time  $T_i$ , such that:

$$\begin{cases} \frac{dY}{dt} = F(Y) = F_K(Y) + F_P(Y), & T_{i-1} < t \leq T_i \\ Y(T_{i-1}) = Y_{i-1}, \\ Q(Y(T_i)) = 0, \end{cases} \quad \text{and } \forall t \in ]T_{i-1}, T_i[, Q(Y(t)) \neq 0 \quad (5.6)$$

where  $F_K(Y)$  and  $F_P(Y)$  are given by 2.34 and 2.35 respectively.

## 5.2 Period of 1 day

Tables 5.2 , 5.3 , 5.4 , 5.5 , 5.6 , 5.7 below summarize the numerical results obtained from applying the APTI, Parareal and Modified Parareal algorithm to compute a satellite's trajectory in a  $J_2$  perturbed motion for a period of 1 day, for a set of 6 initial conditions, using 2,4 and 8 MATLAB workers. Each initial condition is given in terms of the 6 Keplerian orbital elements  $(e_0, a_0, i_0, \omega_0, \Omega_0, M_0)$ .

Table 5.2: Performance results for case 1 initial conditions for 1 day

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>1</b>	<b>1</b>	<b>1</b>
$e_0$	0.1	0.1	0.1
$a_0$ (km)	7300	7300	7300
$i_0$ (degrees)	98	98	98
$\omega_0$ (degrees)	10	10	10
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
N	14	2880	2880
$N_f$	N/A	30	30
$n_s$	4	N/A	N/A
$\tau$ (seconds)	1	1	1
$T_s$ (seconds)	62.2728	18.1651	18.1651
$n_I$	7	2	2
$E_{rel}$	7.85 e-17	0	0
$T_2$ (seconds)	43.3010	17.7960	14.7603
$S_2$	1.44	1.02	1.23
$T_4$ (seconds)	39.3763	9.7590	8.2066
$S_4$	1.58	1.86	2.21
$T_8$ (seconds)	39.6100	5.8486	4.9543
$S_8$	1.57	3.11	3.67



Table 5.3: Performance results for case 2 initial conditions for 1 day

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>2</b>	<b>2</b>	<b>2</b>
$e_0$	0.1	0.1	0.1
$a_0$ (km)	7650	7650	7650
$i_0$ (degrees)	98	98	98
$\omega_0$ (degrees)	10	10	10
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
N	13	2880	2880
$N_f$	N/A	30	30
$n_s$	4	N/A	N/A
$\tau$ (seconds)	1	1	1
$T_s$ (seconds)	61.1379	18.1884	18.1884
$n_I$	6	2	2
$E_{rel}$	1.17 e-15	0	0
$T_2$ (seconds)	41.3354	17.7865	13.8085
$S_2$	1.48	1.02	1.32
$T_4$ (seconds)	38.4897	9.7791	7.6443
$S_4$	1.59	1.86	2.38
$T_8$ (seconds)	39.9636	5.8260	4.6093
$S_8$	1.53	3.12	3.95

Table 5.4: Performance results for case 3 initial conditions for 1 day

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>3</b>	<b>3</b>	<b>3</b>
$e_0$	0.0005	0.0005	0.0005
$a_0$ (km)	7300	7300	7300
$i_0$ (degrees)	98	98	98
$\omega_0$ (degrees)	10	10	10
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
N	14	2880	2880
$N_f$	N/A	30	30
$n_s$	4	N/A	N/A
$\tau$ (seconds)	1	1	1
$T_s$ (seconds)	62.3180	18.2282	18.2282
$n_I$	7	2	2
$E_{rel}$	1.43 e-15	0	0
$T_2$ (seconds)	43.7320	17.7305	14.7159
$S_2$	1.42	1.03	1.24
$T_4$ (seconds)	38.7003	9.9250	7.9245
$S_4$	1.61	1.84	2.30
$T_8$ (seconds)	40.5601	5.8992	4.8085
$S_8$	1.54	3.09	3.79

Table 5.5: Performance results for case 4 initial conditions for 1 day

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>4</b>	<b>4</b>	<b>4</b>
$e_0$	0.15	0.15	0.15
$a_0$ (km)	7300	7300	7300
$i_0$ (degrees)	98	98	98
$\omega_0$ (degrees)	10	10	10
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
N	14	2880	2880
$N_f$	N/A	30	30
$n_s$	4	N/A	N/A
$\tau$ (seconds)	1	1	1
$T_s$ (seconds)	61.7423	18.1846	18.1846
$n_I$	7	2	2
$E_{rel}$	2.22 e-15	0	0
$T_2$ (seconds)	43.3455	18.0201	15.3010
$S_2$	1.42	1.01	1.19
$T_4$ (seconds)	38.3868	9.8367	8.4313
$S_4$	1.61	1.85	2.16
$T_8$ (seconds)	40.3448	5.8484	5.0692
$S_8$	1.53	3.11	3.59

Table 5.6: Performance results for case 5 initial conditions for 1 day

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>5</b>	<b>5</b>	<b>5</b>
$e_0$	0.1	0.1	0.1
$a_0$ (km)	7300	7300	7300
$i_0$ (degrees)	80	80	80
$\omega_0$ (degrees)	10	10	10
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
N	14	2880	2880
$N_f$	N/A	30	30
$n_s$	4	N/A	N/A
$\tau$ (seconds)	1	1	1
$T_s$ (seconds)	62.3974	18.1137	18.1137
$n_I$	7	2	2
$E_{rel}$	2.24 e-15	0	0
$T_2$ (seconds)	43.2047	17.9681	14.9209
$S_2$	1.44	1.01	1.21
$T_4$ (seconds)	38.5543	9.9786	8.2740
$S_4$	1.62	1.82	2.19
$T_8$ (seconds)	39.8357	5.9393	4.9535
$S_8$	1.57	3.05	3.66

Table 5.7: Performance results for case 6 initial conditions for 1 day

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>6</b>	<b>6</b>	<b>6</b>
$e_0$	0.1	0.1	0.1
$a_0$ (km)	7300	7300	7300
$i_0$ (degrees)	98	98	98
$\omega_0$ (degrees)	5	5	5
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
N	14	2880	2880
$N_f$	N/A	30	30
$n_s$	4	N/A	N/A
$\tau$ (seconds)	1	1	1
$T_s$ (seconds)	61.7470	20.2553	20.2553
$n_I$	7	2	2
$E_{rel}$	2.09 e-15	0	0
$T_2$ (seconds)	43.2474	17.7613	14.7149
$S_2$	1.43	1.14	1.38
$T_4$ (seconds)	38.2713	9.9060	8.1910
$S_4$	1.61	2.04	2.47
$T_8$ (seconds)	40.3011	5.8863	4.8748
$S_8$	1.53	3.44	4.16

Table 5.8: Average performance results for a period of 1 day

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
N	14	2880	2880
$n_s$	4	-	-
$n_I$	7	2	2
$E_{rel}$	1.54 e-15	0	0
$S_2$	1.44	1.04	1.26
$S_4$	1.60	1.88	2.29
$S_8$	1.54	3.15	3.80

For a period of 1 day, all 3 algorithms APTI, Parareal and Modified Parareal achieved speed-up versus the sequential process, even when using only 2 workers. Parareal outperforms APTI since it achieves higher speed-up. This is quite expected, as discussed in the previous chapter, since the total time of integration is relatively small. Hence Parareal is at an advantage. On the other hand APTI is at a disadvantage because for this relatively short period of time, the chosen EOS condition required choosing a 14 slices, a very small total number of slices compared to 2880 slices for the Parareal algorithm. Additionally, the APTI algorithm needed to compute 4 slices sequentially before starting the parallel process, hence 30% of the slices were computed sequentially, thus the relatively bad performance of APTI. Moreover, a possible method for improving APTI's performance is by changing the EOS condition from 1 rotation to  $\frac{1}{2}$  or  $\frac{1}{4}$  of a rotation, something we did not try.

We note that the Modified Parareal algorithm outperforms the classic Parareal algorithm since it achieves on average speed-up of 1.21 versus the classic Parareal as shown in table 5.9.

Table 5.9: Speed up of Modified Parareal versus Parareal for a period of 1 day

$n_p$	Speed-up
2	1.21
4	1.22
8	1.21
Average	1.21

### 5.3 Period of 3 days

Table 5.10: Performance results for case 1 initial conditions for 3 days

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>1</b>	<b>1</b>	<b>1</b>
$e_0$	0.1	0.1	0.1
$a_0$ (km)	7300	7300	7300
$i_0$ (degrees)	98	98	98
$\omega_0$ (degrees)	10	10	10
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
$N$	42	2592	2592
$N_f$	N/A	20	20
$n_s$	12	N/A	N/A
$\tau$ (seconds)	5	5	5
$T_s$ (seconds)	38.8944	12.3269	12.3269
$n_I$	4	2	2
$E_{rel}$	3.37 e-05	1.70e-05	1.70e-05
$T_2$ (seconds)	20.7622	12.2523	10.7427
$S_2$	1.87	1.01	1.15
$T_4$ (seconds)	19.11	6.7808	6.1241
$S_4$	2.04	1.82	2.01
$T_8$ (seconds)	26.0099	4.4368	3.8004
$S_8$	1.50	2.78	3.24

Table 5.11: Performance results for case 2 initial conditions for 3 days

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>2</b>	<b>2</b>	<b>2</b>
$e_0$	0.1	0.1	0.1
$a_0$ (km)	7650	7650	7650
$i_0$ (degrees)	98	98	98
$\omega_0$ (degrees)	10	10	10
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
N	39	2592	2592
$N_f$	-	20	20
$n_s$	12	N/A	N/A
$\tau$ (seconds)	5	5	5
$T_s$ (seconds)	38.4149	12.3292	12.3292
$n_I$	4	2	2
$E_{rel}$	2.81 e-05	1.32e-10	1.32e-10
$T_2$ (seconds)	20.4081	11.1546	10.6546
$S_2$	1.88	1.11	1.16
$T_4$ (seconds)	20.0809	6.2827	6.0999
$S_4$	1.91	1.96	2.02
$T_8$ (seconds)	25.5615	4.0353	3.8084
$S_8$	1.50	3.06	3.24



Table 5.12: Average performance results for a period of 3 days

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
N	41	2592	2592
$n_s$	12	-	-
$n_I$	4	2	2
$E_{rel}$	3.09 e-05	8.52 e-06	8.52 e-06
$S_2$	1.88	1.06	1.15
$S_4$	1.97	1.89	2.02
$S_8$	1.50	2.92	3.24

For a period of 3 days, all 3 algorithms APTI, Parareal and Modified Parareal achieve speed-up versus the sequential process. APTI achieves higher speed-up than Parareal when using 2 and 4 workers, while Parareal outperforms APTI while using 8 workers. In this case, APTI is still at a disadvantage because the total number of slices is still relatively small. Additionally, APTI needed to perform 12 slices sequentially before starting the parallel process, hence 30% of the slices were computed sequentially.

We note that the Modified Parareal algorithm outperforms the classic Parareal algorithm since it achieves on average speed-up of 1.1 versus the classic Parareal as shown in table 5.13.

Table 5.13: Speed up of Modified Parareal versus Parareal for a period of 3 days

$n_p$	Speed-up
2	1.09
4	1.07
8	1.11
Average	1.09

## 5.4 Period of 20 days

Table 5.14: Performance results for case 1 initial conditions for 20 days

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>1</b>	<b>1</b>	<b>1</b>
$e_0$	0.1	0.1	0.1
$a_0$ (km)	7300	7300	7300
$i_0$ (degrees)	98	98	98
$\omega_0$ (degrees)	10	10	10
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
N	279	5760	5760
$N_f$	N/A	10	10
$n_s$	35	N/A	N/A
$\tau$ (seconds)	30	30	30
$T_s$ (seconds)	51.2577	14.7394	14.7394
$n_I$	8	21	21
$E_{rel}$	6.27 e-5	3.6 e-05	3.6 e-05
$T_2$ (seconds)	19.2606	137.9646	78.7113
$S_2$	2.66	0.11	0.19
$T_4$ (seconds)	14.8369	79.039	45.3034
$S_4$	3.45	0.19	0.33
$T_8$ (seconds)	15.6309	51.5848	29.969
$S_8$	3.28	0.29	0.49

Table 5.15: Performance results for case 2 initial conditions for 20 day

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>2</b>	<b>2</b>	<b>2</b>
$e_0$	0.1	0.1	0.1
$a_0$ (km)	7650	7650	7650
$i_0$ (degrees)	98	98	98
$\omega_0$ (degrees)	10	10	10
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
N	260	5760	5760
$N_f$	N/A	10	10
$n_s$	35	N/A	N/A
$\tau$ (seconds)	30	30	30
$T_s$ (seconds)	50.6646	13.755	13.755
$n_I$	7	17	17
$E_{rel}$	5.33 e-05	4.07 e-05	4.07 e-05
$T_2$ (seconds)	19.9946	110.1763	64.5697
$S_2$	<b>2.53</b>	<b>0.12</b>	<b>0.21</b>
$T_4$ (seconds)	14.5221	64.2077	37.6062
$S_4$	<b>3.49</b>	<b>0.21</b>	<b>0.37</b>
$T_8$ (seconds)	15.6356	42.0768	24.9649
$S_8$	<b>3.24</b>	<b>0.33</b>	<b>0.55</b>

Table 5.16: Average performance results for a period of 20 days

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
N	270	5760	5760
$n_s$	35	-	-
$n_I$	8	19	19
$E_{rel}$	5.8 e-05	3.84 e-05	3.84 e-05
$S_2$	2.60	0.12	0.20
$S_4$	3.47	0.20	0.35
$S_8$	3.26	0.31	0.52

For a period of 20 days, only APTI was able to achieve a speed-up versus the sequential process, while Parareal and Modified Parareal fail to do so. APTI outperforms Parareal since the total time of integration is relatively large, which enforced a reasonable total number of slices for APTI.

One must also note that APTI has faster convergence rate since it required 8 iterations to converge while Parareal needed 19 iterations.

Moreover, the Modified Parareal algorithm outperforms the classic Parareal algorithm since it takes on average half of the execution time of the classic Parareal, hence achieving almost twice the speed-up as shown in table 5.17.

We do note that the speed-up for Parareal increased as the number of workers increased, but we could not test it with more workers since the AUB HPC platform had a licence for very few MATLAB workers.

Table 5.17: Speed up of Modified Parareal versus Parareal for a period of 20 days

$n_p$	Speed-up
2	1.73
4	1.73
8	1.7
Average	1.72

## 5.5 Period of 43 days

Table 5.18: Performance results for case 1 initial conditions for 43 days

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>1</b>	<b>1</b>	<b>1</b>
$e_0$	0.1	0.1	0.1
$a_0$ (km)	7300	7300	7300
$i_0$ (degrees)	98	98	98
$\omega_0$ (degrees)	10	10	10
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
N	599	12384	12384
$N_f$	N/A	5	5
$n_s$	46	N/A	N/A
$\tau$ (seconds)	60	60	60
$T_s$ (seconds)	66.763	14.9342	14.9342
$n_I$	10	42	42
$E_{rel}$	1.04 e-04	1.3 e-04	1.3 e-04
$T_2$ (seconds)	23.2285	336.3072	177.3699
$S_2$	2.87	0.04	0.08
$T_4$ (seconds)	15.5201	211.1520	111.0128
$S_4$	4.30	0.07	0.13
$T_8$ (seconds)	13.8165	153.9589	81.2850
$S_8$	4.83	0.10	0.18

Table 5.19: Performance results for case 2 initial conditions for 43 days

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
<b>Case</b>	<b>2</b>	<b>2</b>	<b>2</b>
$e_0$	0.1	0.1	0.1
$a_0$ (km)	7650	7650	7650
$i_0$ (degrees)	98	98	98
$\omega_0$ (degrees)	10	10	10
$\Omega_0$ (degrees)	45	45	45
$M_0$ (degrees)	123	123	123
N	558	12384	12384
$N_f$	N/A	5	5
$n_s$	46	N/A	N/A
$\tau$ (seconds)	60	60	60
$T_s$ (seconds)	64.7477	14.8966	14.8966
$n_I$	9	33	33
$E_{rel}$	8.29 e-05	2.01 e-04	2.01 e-04
$T_2$ (seconds)	22.7323	264.8994	141.4436
$S_2$	<b>2.85</b>	<b>0.06</b>	<b>0.11</b>
$T_4$ (seconds)	15.1349	167.8680	89.8344
$S_4$	<b>4.28</b>	<b>0.09</b>	<b>0.17</b>
$T_8$ (seconds)	13.3199	121.5371	65.7219
$S_8$	<b>4.86</b>	<b>0.12</b>	<b>0.23</b>

Table 5.20: Average performance results for a period of 43 days

	<b>APTI</b>	<b>Parareal</b>	<b>Modified Parareal</b>
N	579	12384	12384
$n_s$	46	-	-
$n_I$	10	38	38
$E_{rel}$	9.34 e-05	1.65 e-04	1.65 e-04
$S_2$	2.86	0.05	0.09
$S_4$	4.29	0.08	0.15
$S_8$	4.85	0.11	0.21

For a period of 43 days, APTI algorithm is also the only one to achieve speed-up versus the sequential process.

Furthermore, APTI has faster convergence rate since it required 10 iterations to converge while Parareal needed 38 iterations. Hence APTI's convergence rate is 4 times better than that of Parareal.

Moreover, the Modified Parareal algorithm outperforms the classic Parareal algorithm since it takes on average half of the execution time of the classic Parareal, hence achieving almost twice the speed-up as shown in table 5.21.

Table 5.21: Speed up of Modified Parareal versus Parareal for a period of 43 days

$n_p$	Speed-up
2	1.89
4	1.89
8	1.89
Average	1.89

## 5.6 Period of 108 days

Table 5.22: Performance results (1) of APTI for 108 days

Case	1	2	3	4
$e_0$	0.1	0.0005	0.15	0.1
$a_0$ (km)	7300	7300	7300	7300
$i_0$ (degrees)	98	98	98	98
$\omega_0$ (degrees)	10	10	10	5
$\Omega_0$ (degrees)	45	45	45	45
$M_0$ (degrees)	123	123	123	123
N	1500	1500	1500	1500
$n_s$	46	46	46	46
$\tau$ (seconds)	60	60	60	60
$T_s$ (seconds)	166.2018	167.7576	167.463	166.5701
$n_I$	18	17	22	29
$E_{rel}$	2.94 e-03	1.50 e-03	2.06 e-02	5.90 e-03
$T_2$ (seconds)	52.9642	52.3221	53.8457	54.1518
$S_2$	3.14	3.21	3.11	3.08
$T_4$ (seconds)	32.8032	31.7628	33.4829	34.219
$S_4$	5.07	5.28	5.00	4.87
$T_8$ (seconds)	26.336	25.3389	28.6234	30.616
$S_8$	6.31	6.62	5.85	5.44



Table 5.23: Performance results (2) of APTI for 108 days

<b>Case</b>	5	6	7	8
$e_0$	0.1	0.1	0.1	0.1
$a_0$ (km)	7300	7300	7300	7300
$i_0$ (degrees)	98	98	98	98
$\omega_0$ (degrees)	10	10	10	10
$\Omega_0$ (degrees)	10	120	45	45
$M_0$ (degrees)	123	123	20	60
N	1500	1500	1500	1500
$n_s$	46	46	46	46
$\tau$ (seconds)	60	60	60	60
$T_s$ (seconds)	166.5804	166.6014	166.7929	167.4185
$n_I$	18	18	17	26
$E_{rel}$	2.57 e-03	2.57 e-03	3.23 e-03	8.88 e-03
$T_2$ (seconds)	52.9488	52.4362	52.6763	53.5668
$S_2$	<b>3.15</b>	<b>3.18</b>	<b>3.17</b>	<b>3.13</b>
$T_4$ (seconds)	32.1936	31.7623	31.5856	33.1685
$S_4$	<b>5.17</b>	<b>5.25</b>	<b>5.28</b>	<b>5.05</b>
$T_8$ (seconds)	26.4352	26.1684	25.3772	27.4835
$S_8$	<b>6.30</b>	<b>6.37</b>	<b>6.57</b>	<b>6.09</b>

Table 5.24: Average performance results of APTI for a period of 108 days

$N$	1500
$n_s$	46
$n_I$	21
$E_{rel}$	6.03 e-03
$S_2$	3.14
$S_4$	5.12
$S_8$	6.19

For a period of 108 days, we evaluated APTI on a set of 8 initial conditions. Table 5.24 shows the average performance results obtained. The results are very promising since:

- The algorithm shows a fast convergence rate, because it converges after 21 iterations which is negligible when compared to the total number of slices 1500.
- The speed-up is significant while using 2,4 and 8 workers.

# Chapter 6

## Conclusion

The time-dependent ordinary differential equations that describe the motion of a satellite in a  $J_2$  perturbed potential, can in fact be solved in a time-parallel manner using predictor-corrector schemes. In this thesis, we presented two main methods in order to do so: the Adaptive Parallel Time Integration algorithm (APTI) and the Parareal algorithm. Each algorithm has its own advantages and disadvantages. APTI has the advantage of yielding good predictions because it solves the first  $n_s$  slices sequentially then uses this exact solution of the problem in order to make predictions for the rest of slices. APTI also uses ratio-based corrections which are more accurate than other correction methods. But the disadvantage of the APTI algorithm is that it involves the sequential computation of  $n_s$  slices which is sometimes costly, and the ratio-based correction is relatively slow which can adversely affect execution time.

On the other hand Parareal has the advantage of not solving any slices sequentially, which saves on execution time, and it uses a relatively fast correction scheme which consists on propagating the jumps. But on the downside, Parareal's predictions are not as accurate as APTI's and they are also dependent on all the slices. Additionally, Parareal's corrections involve lots of communications since they are also dependent.

We tried to improve Parareal's performance by introducing the modified Parareal algorithm. This algorithm intends to save on execution time by

assessing the convergence of every slice individually during every iteration, then not computing the already converged slices during the following iteration of the algorithm.

The results have shown that the Parareal algorithm outperforms the APTI algorithm for relatively small integration periods, since the chosen EOS condition enforced choosing a relatively small number of slices. The small total number of slices proved to be not significant enough to get a valuable speed-up.

On the other hand, APTI outperforms Parareal (in terms of execution time and number of iterations) for relatively large integration periods, when the total number of slices became much larger. Thus we showed APTI's efficiency for predicting the orbit of a satellite in a  $J_2$  perturbed motion, for up to 108 days with an average relative error of  $6.10^{-3}$  which is very insignificant considering the total integration time  $T = 9,331,200$  seconds.

Moreover, the results also show that the modified Parareal algorithm takes less execution time than the classic Parareal algorithm for small and large integration periods, hence doubling the speed-up.

As future work, we plan the following:

- Change the EOS condition for APTI from doing 1 whole iteration about the xy-plane in the IPQW frame into doing  $\frac{1}{2}$  or  $\frac{1}{4}$  of a rotation, in order to increase the total number of slices for small integration periods. This will help improve APTI's performance for small time intervals.
- Increase the number of MATLAB workers by getting a license for a bigger number of workers, then test if Parareal's execution time will decrease for relatively large integration periods.
- Compare our numerical results with analytical ones given by classical perturbation theory.
- Modify the Parareal algorithm in such a way that the coarse slices are no longer equal, but take the same execution time. For instance, this could be done by first doing trial run and determining which coarse partition is taking the longest amount of execution time, and which one is taking the shortest amount of execution time. Then one would adjust the length of these partitions so that their respective execution

times become equal. This approach is extremely useful to avoid any idle time on any processor.

# Bibliography

- [1] Pierluigi Amodio and Luigi Brugnano. Parallel solution in time of odes: some achievements and perspectives. *Applied Numerical Mathematics*, 59:424 – 435, 2009.
- [2] Leonardo Baffico, Stephane Bernard, Yvon Maday, Gabriel Turinici, and Gilles Zérah. Parallel-in-time molecular-dynamics simulations. *Physical Review E*, 66(5):057701, 2002.
- [3] Guillaume Bal. On the convergence and the stability of the parareal algorithm to solve partial differential equations. In *Domain decomposition methods in science and engineering*, pages 425–432. Springer, 2005.
- [4] Guillaume Bal and Yvon Maday. A “parareal” time discretization for non-linear pde’s with application to the pricing of an american put. In *Recent developments in domain decomposition methods*, pages 189–202. Springer, 2002.
- [5] Kevin Burrage. Parallel methods for initial value problems. *Applied Numerical Mathematics*, 11(1):5–25, 1993.
- [6] Kevin Burrage. Parallel methods for odes. *Advances in Computational Mathematics*, 7(1-2):1–31, 1997.
- [7] P. Chartier and B. Philippe. A parallel shooting technique for solving dissipative ode’s. *Computing*, 51(3-4):209–236, 1993.
- [8] Philippe Chartier and Bernard Philippe. A parallel shooting technique for solving dissipative ode’s. *Computing*, 51(3-4):209–236, 1993.

- [9] Jocelyne Erhel and Stéphanie Rault. Algorithme parallèle pour le calcul d'orbites. 1998.
- [10] Jocelyne Erhel and Stéphanie Rault. Algorithme parallèle pour le calcul d'orbites: Parallélisation à travers le temps. *TSI. Technique et science informatiques*, 19(5):649–673, 2000.
- [11] Charbel Farhat and Marion Chandesris. Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid–structure applications. *International Journal for Numerical Methods in Engineering*, 58(9):1397–1434, 2003.
- [12] Charbel Farhat, Julien Cortial, ClimÁlne Dastillung, and Henri Bavestrello. Time-parallel implicit integrators for the near-real-time prediction of linear structural dynamic responses. *International Journal for Numerical Methods in Engineering*, 67(5):697–724, 2006.
- [13] Paul F Fischer, Frédéric Hecht, and Yvon Maday. A parareal in time semi-implicit approximation of the navier-stokes equations. In *Domain decomposition methods in science and engineering*, pages 433–440. Springer, 2005.
- [14] Martin J Gander and Ernst Hairer. Nonlinear convergence analysis for the parareal algorithm. In *Domain decomposition methods in science and engineering XVII*, pages 45–56. Springer, 2008.
- [15] Martin J Gander and Stefan Vandewalle. Analysis of the parareal time-parallel time-integration method. *SIAM Journal on Scientific Computing*, 29(2):556–578, 2007.
- [16] Martin J Gander and Stefan Vandewalle. On the superlinear and linear convergence of the parareal algorithm. In *Domain decomposition methods in science and engineering XVI*, pages 291–298. Springer, 2007.
- [17] Izaskun Garrido, Magne S Espedal, and Gunnar E Fladmark. A convergent algorithm for time parallelization applied to reservoir simulation. In *Domain Decomposition Methods in Science and Engineering*, pages 469–476. Springer, 2005.
- [18] C.W. Gear. Parallel methods for ordinary differential equations. *CAL-COLO*, 25(1-2):1–20, 1988.

- [19] Herbert Goldstein. *Classical Mechanics*. Addison-Wesley Publishing Company, Reading, MA, 2nd edition, 1980.
- [20] J.Haykal. The rapti algorithm and its applications to time-parallel numerical integration. Master's thesis, AUB, June 2009.
- [21] Noha Makhoul Karam. *Time-Slicing, Rescaling and Ratio-based Parallel Time Integration*. PhD thesis, Université de Rennes 1, 2010.
- [22] Noha Makhoul Karam, Nabil Nassif, and Jocelyne Erhel. An adaptive parallel-in-time method with application to a membrane problem. 2013.
- [23] Jacques-Louis Lions, Yvon Maday, and Gabriel Turinici. Résolution d'edp par un schéma en temps «pararéel». *Comptes Rendus de l'Académie des Sciences-Series I-Mathematics*, 332(7):661–668, 2001.
- [24] Yvon Maday and Gabriel Turinici. A parallel in time approach for quantum control: the parareal algorithm. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 1, pages 62–66. IEEE, 2002.
- [25] Yvon Maday and Gabriel Turinici. The parareal in time iterative solver: a further direction to parallel implementation. In *Domain decomposition methods in science and engineering*, pages 441–448. Springer, 2005.
- [26] Nabil R. Nassif, Dolly Fayyad, and Maria Cortas. Sliced-time computations with re-scaling for blowing-up solutions to initial value differential equations. In VaidyS. Sunderam, GeertDick Albada, PeterM.A. Sloot, and JackJ. Dongarra, editors, *Computational Science & ICCS 2005*, volume 3514 of *Lecture Notes in Computer Science*, pages 58–65. Springer Berlin Heidelberg, 2005.
- [27] Nabil R Nassif, Noha Makhoul Karam, and Yeran Soukiassian. A new approach for solving evolution problems in time-parallel way. In *Computational Science-ICCS 2006*, pages 148–155. Springer, 2006.
- [28] Nabil R Nassif, Noha Makhoul-Karam, and Yeran Soukiassian. Computation of blowing-up solutions for second-order differential equations using re-scaling techniques. *Journal of Computational and Applied Mathematics*, 227(1):185–195, 2009.



- [29] J. Nievergelt. Parallel methods for integration ordinary differential equations. *Comm. ACM*, 7:731–733, 1964.
- [30] Gunnar Staff. Convergence and stability of the parareal algorithm: A numerical and theoretical investigation. *preprint Numerics*, (2), 2003.
- [31] Gunnar A Staff. The parareal algorithm, 2003.
- [32] Gunnar A Staff. Solving the unsteady navier-stokes equation an introduction to how to handle time. 2004.
- [33] Y.Soukiassian. Parallel algorithms for time integration. Master's thesis, AUB, September 2007.
- [34] Olivier Zarrouati. Trajectoires spatiales. *Trajectoires spatiales.. O. Zarrouati. Cepadues-Editions, 111 rue Nicholas-Vauquelin, F-31100 Toulouse, France. 522 pp. Price FF 199.00 (1987). ISBN 2-85428-166-7.*, 1, 1987.